A MARSHALL CAVENDISH **29** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING – FOR FUN AND THE FUTURE



UK £1.00 Republic of Ireland £1.25 Malta 85c Australia $2.25 New Zealand $2.95

INPUT

Vol. 3 **No 29**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

IMPORTANT NOTICE

The **Cliffhanger** listings published in this and subsequent issues of INPUT bear no resemblance to and are in no way associated with the computer game called **Cliff Hanger** owned by New Generation Software Ltd.

PICTURE CREDITS

Front cover, Dave King. Pages 889, 890, 895, Kate Charlesworth. Pages 889, 891, 893, 895, Peter Reilly. Page 896, Phil Dobson. Pages 889, 900, 901, 903, Dave King. Pages 904, 907, 909, 910, 912, 913, Paul Davies. Pages 914, 916, 918, Dave King.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd,
Department 980, Newtown Road,
Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services,
Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd,
Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

LOOKING INTO CURVES

- RECOGNIZING THE CURVES
- FALLING DOWN AN ELLIPSE
- A PARABOLIC SWIM
- CIRCLES AND POLYGONS
- GRAPHICS FROM CURVES

The first article on conic sections showed how to draw a circle, ellipse, parabola and hyperbola. This time you'll see how to incorporate them into your own programs

All these conic curves crop up in everyday life, often in unexpected ways, and a few examples were given last time.

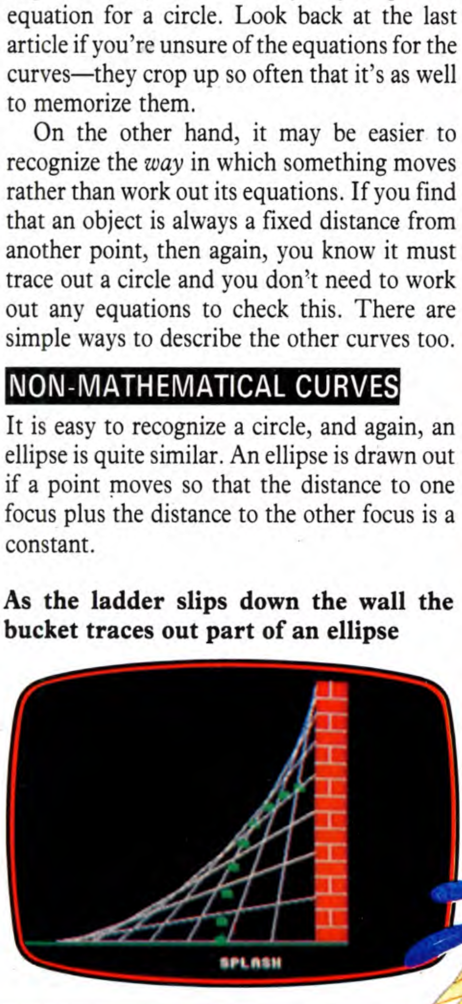
The trick, really, is in recognizing when the equations for one of the curves apply. Sometimes it is easy. If you work out the position of a moving object, or a point on a line and you find that its X coordinate is given by A*COS T and its Y coordinate is given by A*SIN T (where A is a fixed distance and angle T varies) then it is fairly easy to spot the equation for a circle. Look back at the last article if you're unsure of the equations for the curves—they crop up so often that it's as well to memorize them.

On the other hand, it may be easier to recognize the *way* in which something moves rather than work out its equations. If you find that an object is always a fixed distance from another point, then again, you know it must trace out a circle and you don't need to work out any equations to check this. There are simple ways to describe the other curves too.

NON-MATHEMATICAL CURVES

It is easy to recognize a circle, and again, an ellipse is quite similar. An ellipse is drawn out if a point moves so that the distance to one focus plus the distance to the other focus is a constant.

As the ladder slips down the wall the bucket traces out part of an ellipse





A parabola is traced out if a point moves so that the distance to a fixed point is the same as the perpendicular distance to a fixed line. The fixed point becomes the focus of the parabola and the line becomes the *directrix* which is a line at right angles to the axis, outside the parabola, and the same distance from the curve as the focus.

The hyperbola is simpler to describe, and is drawn when a point moves so the distance to one fixed point *minus* the distance to another fixed point is a constant. The fixed points become the foci of the hyperbola. Both foci are needed to create the hyperbola, one in each half of the curve, which is why it is not strictly accurate to call just one half of the curve an hyperbola.

The programs below demonstrate both methods of recognizing the curves, either by spotting the equations or by noticing the way a point or object moves. Once you've spotted one of the curves in your own programs you'll be able to deal with them much more easily.

SLIPPING LADDER

The first program shows how an ellipse is connected with such ordinary events as a ladder slipping down a wall. You'll find that a bucket attached to the ladder traces out part of an ellipse as the ladder slips down. As an example, if the ladder is 800 units long with the bucket hung on a rung 500 units from the bottom. The position of the bucket is X = −3ØØ*COS(angle) and Y = 5ØØ*SIN (angle) which you should recognize by now as the equation of an ellipse.

```
1Ø LET wall=24Ø: LET ladder=6Ø: LET
   bucket=19Ø
2Ø GOSUB wall
3Ø GOSUB ladder
35 FLASH Ø
4Ø GOTO 4Ø
6Ø FOR a=8Ø TO Ø STEP −1Ø
7Ø PAUSE 25: LET r=a/(18Ø/PI)
8Ø PLOT ox−15Ø*COS (r),oy
9Ø DRAW ox−(ox−15Ø*COS (r)),
   oy+15Ø*SIN (r)
11Ø LET x=−6Ø*COS (r)
12Ø LET y=9Ø*SIN (r)
13Ø GOSUB bucket
14Ø BEEP .1,a/2−15
15Ø NEXT a
16Ø FLASH 1: PRINT AT 1Ø,5;"SPLASH"
17Ø RETURN
19Ø PLOT ox+x,oy+y+5:DRAW Ø,−2
2ØØ FOR n=oy+y TO oy+y+2: PLOT
   ox+x−2,n: DRAW 4,Ø
21Ø NEXT n
22Ø RETURN
24Ø BORDER Ø: INK 7: PAPER Ø: CLS
25Ø LET ox=232: LET oy=8
26Ø FOR y=1 TO 2Ø: PRINT PAPER 2;AT
   y,29;"□□"
27Ø NEXT y
28Ø FOR y=oy−1 TO 165 STEP 16:
   PLOT;ox,y
29Ø DRAW 16,Ø: PLOT ox,y+8: DRAW 16,Ø:
   PLOT ox+8,y+8: DRAW Ø,8
3ØØ NEXT y
31Ø PLOT INK 4;ox+8,oy−1: DRAW INK
   4;−232,Ø
32Ø RETURN
```

```
1Ø HIRES 1,6:MULTI 2,6,5:
   COLOUR 1,3:C=ATN(1)/45
2Ø GOSUB 23Ø
3Ø GOSUB 5Ø
4Ø GOTO 4Ø
5Ø FOR AN=8Ø TO Ø STEP −1Ø
9Ø LINE 115−75*COS(C*AN),155,114,
   15Ø−15Ø*SIN(C*AN),1
11Ø X=−28*COS(C*AN)
12Ø Y=9Ø*SIN(C*AN)
13Ø GOSUB 2ØØ
14Ø FOR T=Ø TO 2ØØ:NEXT T
15Ø NEXT AN
16Ø RETURN
2ØØ IF Y=Ø THEN Y=4:TEXT Ø,16Ø,
   "SPLASH",Ø,5,3Ø
21Ø TEXT 114+X,154−Y,"◘",2,1,8
22Ø RETURN
23Ø BLOCK 115,Ø,127,15Ø,1
25Ø FOR Y=Ø TO 14Ø STEP 1Ø
26Ø LINE 115,Y,127,Y,Ø
27Ø NEXT Y
28Ø FOR Y=Ø TO 15Ø STEP 2Ø
29Ø LINE 121,Y,121,Y+1Ø,Ø
3ØØ NEXT Y
31Ø BLOCK Ø,151,16Ø,199,3
32Ø RETURN
```

```
1Ø GRAPHIC 1: COLOR 1,6,2,5:
   C=ATN(1)/45
2Ø GOSUB 23Ø
3Ø GOSUB 5Ø
4Ø GOTO 4Ø
5Ø FOR AN=8Ø TO Ø STEP −1Ø
9Ø DRAW 1,1Ø23−45Ø*COS(C*AN),775 TO
   1Ø23,75Ø−75Ø*SIN(C*AN)
11Ø X=−112*COS(C*AN)
12Ø Y=45Ø*SIN(C*AN)
13Ø GOSUB 2ØØ
14Ø FOR T=Ø TO 2ØØ:NEXT T
15Ø NEXT AN
16Ø RETURN
2ØØ IF Y=Ø THEN Y=4:CHAR 7,7,
   "SPLASH"
21Ø CIRCLE 3,1Ø23+X,77Ø−Y,1Ø,1Ø
22Ø RETURN
23Ø DRAW 2,1Ø23,Ø TO 1Ø23,8ØØ
31Ø FOR Z=1 TO 24Ø STEP 1Ø: DRAW
   3,Ø,775+Z TO 1Ø23,775+Z:NEXT Z
32Ø RETURN
```

```
1Ø MODE 1
2Ø PROCWall
3Ø PROCLadder
4Ø END
5Ø DEF PROCLadder
6Ø FOR angle=8Ø TO Ø STEP −1Ø
7Ø GCOLØ,3
8Ø MOVE −8ØØ*COS(RAD(angle)),Ø
9Ø DRAW Ø,8ØØ*SIN(RAD(angle))
1ØØ VDU19,2,2;Ø;:GCOLØ,2
11Ø x=−3ØØ*COS(RAD(angle))
12Ø y=5ØØ*SIN(RAD(angle))
13Ø PROCBucket(x,y)
14Ø FOR T=Ø TO 1ØØØ:NEXT
```



```
15Ø NEXT angle
16Ø MOVE -1Ø1Ø,-21Ø
17Ø ENDPROC
18Ø DEF PROCBucket(x,y)
19Ø VDU23,24Ø,8,8,8,255,255,255,255,255
2ØØ IF y=Ø THEN y=3Ø:MOVE -3ØØ, -5Ø:
    GCOLØ,3:PRINT"SPLASH":GCOLØ,2
21Ø VDU5:MOVE x-1Ø,y:VDU 24Ø
22Ø ENDPROC
23Ø DEF PROCWall
24Ø VDU29,1ØØØ;2ØØ;
25Ø GCOLØ,1:MOVE Ø,Ø:MOVE 1ØØ,Ø
26Ø PLOT85,1ØØ,8ØØ:MOVE Ø,8ØØ
27Ø PLOT85,Ø,Ø
28Ø GCOLØ,3
29Ø FOR Y=Ø TO 7ØØ STEP 1ØØ
3ØØ MOVE Ø,Y:DRAW 1ØØ,Y
31Ø MOVE Ø,Y+5Ø:DRAW 1ØØ,Y+5Ø
32Ø MOVE 5Ø,Y+5Ø:DRAW 5Ø,Y+1ØØ
33Ø NEXT Y
34Ø GCOLØ,2:MOVE 1ØØ,-4:DRAW
    -1ØØØ,-4
35Ø ENDPROC
```

```
1Ø PMODE3,1:PCLS:SCREEN1,Ø:
    C=ATN(1)/45
2Ø GOSUB23Ø
3Ø GOSUB5Ø
4Ø GOTO4Ø
5Ø FORAN=8Ø TO Ø STEP -1Ø
7Ø COLOR4,2
9Ø LINE(23Ø-15Ø*COS(C*AN),15Ø)-
    (228,15Ø-15Ø*SIN(C*AN)),PRESET
11Ø X=-56*COS(C*AN)
12Ø Y=9Ø*SIN(C*AN)
13Ø GOSUB2ØØ
14Ø FORT=ØTO5ØØ:NEXT
15Ø NEXT
16Ø RETURN
2ØØ IF Y=Ø THENY=4:DRAW"BM16Ø,
    156C2S16LDRDLBR2U2RDLBEBRD2RB
    RU2RDNLDBRRULURBRD
    2BRUNLUC4"
21Ø LINE(228+X,154-Y)-(232+X,
    15Ø-Y),PSET,BF
22Ø RETURN
23Ø LINE(23Ø,Ø)-(255,15Ø),PSET,BF
24Ø COLOR2
25Ø FORY=ØTO15Ø STEP1Ø
26Ø LINE(23Ø,Y)-(255,Y),PSET
27Ø NEXT
28Ø FORY=ØTO15Ø STEP2Ø
29Ø LINE(243,Y)-(243,Y+1Ø),PSET
3ØØ NEXT
31Ø COLOR3:LINE(Ø,151)-(255,191),
    PSET,BF
32Ø RETURN
```



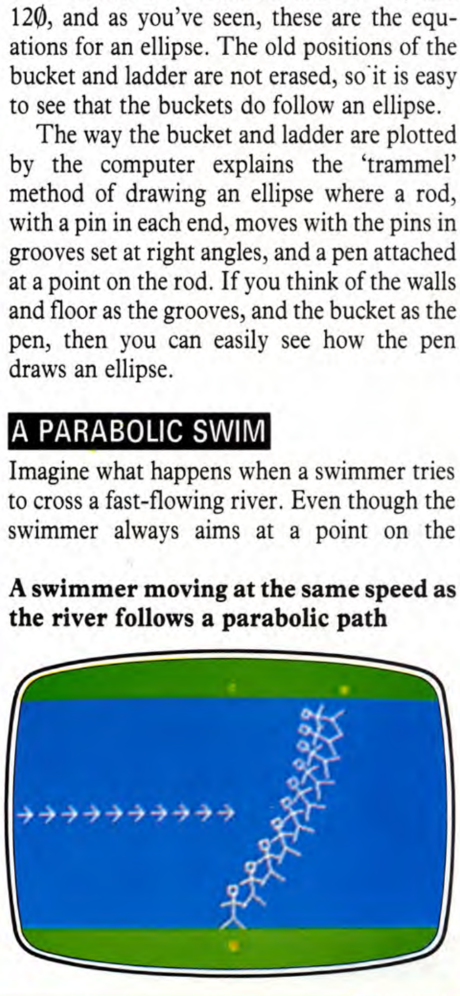
The program consists of three main routines—to draw the wall, the ladder and the bucket. The wall is drawn first by Lines 23Ø to 35Ø, then the routine at Lines 5Ø to 17Ø draws the ladder in nine different positions at intervals of 10° as it slips down the wall. This routine also calls the bucket drawing routine at Lines 18Ø to 22Ø to draw in the bucket for each position of the ladder. The coordinates of the bucket are worked out at Lines 11Ø and 12Ø, and as you've seen, these are the equations for an ellipse. The old positions of the bucket and ladder are not erased, so it is easy to see that the buckets do follow an ellipse.

The way the bucket and ladder are plotted by the computer explains the 'trammel' method of drawing an ellipse where a rod, with a pin in each end, moves with the pins in grooves set at right angles, and a pen attached at a point on the rod. If you think of the walls and floor as the grooves, and the bucket as the pen, then you can easily see how the pen draws an ellipse.

A PARABOLIC SWIM

Imagine what happens when a swimmer tries to cross a fast-flowing river. Even though the swimmer always aims at a point on the

A swimmer moving at the same speed as the river follows a parabolic path



opposite bank the current actually carries him downstream a certain distance. If the river flows at the same speed as the swimmer then the distance downstream equals half the width of the river, and the combined effect of the current and the swimmer's own speed carries him on a parabolic path.

To understand why this happens you have to think in terms of velocities. The swimmer always aims at the point on the bank with velocity V and the river flows parallel to the bank with the same velocity V. You can combine these two velocities to give the swimmers actual velocity relative to the bank.

This uses the parallelogram of forces which, if you've forgotten about or don't know about, you'll have to take on trust. But exactly the same diagram is used to construct a parabola where the distance the swimmer moves to the focus (in this case the point on the bank) equals the distance to the directrix (which is the distance the river flows in the same time).

```
10 BORDER Ø: PAPER Ø: INK 7: CLS
50 REM DRAW RIVER
60 LET parabola=19Ø: LET swimmer=3ØØ:
   LET rotate=43Ø
70 LET a$="□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□"
80 FOR n=Ø TO 3
90 PRINT PAPER 4;a$: NEXT n
100 FOR n=4 TO 18
110 PRINT PAPER 1;a$: NEXT n
120 FOR n=19 TO 21
130 PRINT PAPER 4;a$: NEXT n
140 PRINT PAPER 2; INK 6;AT 3,15;"F";AT
   19,15;"A";AT 3,22;"O"
150 GOSUB parabola
160 STOP
190 LET ox=187: LET oy=15Ø
200 FOR t=-1 TO -Ø.Ø5 STEP Ø.1
210 LET x=-6Ø*(t*t): LET y=12Ø*t
220 LET a=ATN ((x+6Ø)/-y)
230 PLOT ox-6Ø+y,oy-6Ø: DRAW INK
   7;1Ø,Ø: DRAW INK 7;-5,-5: DRAW INK
   7;Ø,1Ø: DRAW INK 7;5,-5
240 GOSUB swimmer
250 NEXT t
260 RETURN
300 LET ox=ox+x: LET oy=oy+y
310 LET x=Ø: LET y=6
320 GOSUB rotate
330 PLOT ox+xt,oy+yt
340 RESTORE 41Ø
350 FOR n=1 TO 17
360 READ x,y
370 GOSUB rotate
380 DRAW xt,yt
390 NEXT n
400 LET ox=188: LET oy=15Ø: RETURN
410 DATA -3,Ø,Ø,3,3,Ø,Ø,-3,-2,Ø,Ø,-3,
   -4,Ø,Ø,4,Ø,-4,8,Ø,Ø,4,Ø,-4,
   -4,Ø,Ø,-4,-4,-4,4,4,4,-4
430 LET xt=x*COS (a)-y*SIN (a)
440 LET yt=x*SIN (a)+y*COS (a)
450 RETURN
```

```
10 HIRES 1,6:COLOUR 6,1
20 GOSUB 5Ø
30 GOSUB 2ØØ
40 GOTO 4Ø
50 BLOCK Ø,38,319,153,1
60 TEXT 123,158,"A",1,1,8
70 TEXT 123,25,"F",1,1,8
80 TEXT 18Ø,25,"O",1,1,8
90 RETURN
200 FOR T=-1 TO -.Ø5 STEP .1
210 X=-6Ø*T*T:Y=12Ø*T
220 AN=ATN((X+6Ø)/-Y)
230 GOSUB 3ØØ
240 TEXT 13Ø+Y,95,">",Ø,1,8
250 NEXT T
260 RETURN
300 XC=187+X:YC=33-Y
310 XX=Ø:YY=6:GOSUB 43Ø
320 X1=XC+XT:Y1=YC-YT
330 RESTORE
340 FOR N=1 TO 16
350 READ XX,YY
360 GOSUB 43Ø
370 LINE X1,Y1,XC+XT+.5,
   YC-YT+.5,Ø
375 X1=XC+XT+.5:Y1=YC-YT+.5
380 NEXT N
390 RETURN
410 DATA Ø,6,-6,Ø,Ø,6,6,Ø,Ø,6,Ø,14,6,12,
   Ø,14,-6,12,Ø,14,Ø,18,2,18
415 DATA 2,22,-2,22,-2,18,Ø,18
430 XT=XX*COS(AN)-YY*SIN(AN)
440 YT=XX*SIN(AN)+YY*COS(AN)
450 RETURN
```

```
10 GRAPHIC 2:COLOR 6,2,5,5
20 GOSUB 5Ø
30 GOSUB 2ØØ
40 GOTO 4Ø
50 DRAW 1,Ø,12Ø TO 1Ø23,11Ø:
   PAINT 1,Ø,Ø
55 DRAW 1,Ø,8Ø8 TO 1Ø23,8ØØ:
   PAINT 1,Ø,1Ø23
60 CHAR Ø,14,"A"
70 CHAR Ø,8,"F"
80 CHAR 17,8,"O"
90 RETURN
200 FOR T=-1 TO -.Ø5 STEP .1
210 X=-34Ø*T*T:Y=6ØØ*T
220 AN=ATN((X+6Ø)/-Y)
230 GOSUB 3ØØ
240 S=S+1:CHAR 8,S,"W"
250 NEXT T
260 RETURN
300 XC=748+X:YC=165-Y
310 XX=Ø:YY=6:GOSUB 43Ø
320 POINT Ø,XC+XT,YC-YT
330 RESTORE
340 FOR N=1 TO 16
350 READ XX,YY:XX=XX*4:YY=YY*4
360 GOSUB 43Ø
370 DRAW 1 TO XC+XT+.5,YC-YT+.5
380 NEXT N
390 RETURN
410 DATA Ø,6,-6,Ø,Ø,6,6,Ø,Ø,6,Ø,14,6,12,
   Ø,14,-6,12,Ø,14,Ø,18,2,18
415 DATA 2,22,-2,22,-2,18,Ø,18
430 XT=XX*COS(AN)-YY*SIN(AN)
440 YT=XX*SIN(AN)+YY*COS(AN)
450 RETURN
```

```
10 MODE 1
20 PROCRiver
30 PROCParabola
40 END
50 DEF PROCRiver
60 VDU19,Ø,4;Ø;
70 VDU19,2,2;Ø;:GCOLØ,2
80 MOVE Ø,812:MOVE 1279,812
90 PLOT 85,1279,1Ø23:MOVE Ø,1Ø23
100 PLOT 85,Ø,812:MOVE Ø,Ø
110 MOVE 1279,Ø:PLOT85,1279,212
120 MOVE Ø,212:PLOT85,Ø,Ø
130 GCOLØ,1:VDU5:MOVE 63Ø,18Ø
140 PRINT"A":MOVE 63Ø,85Ø:PRINT"F"
150 MOVE 93Ø,85Ø:PRINT"O"
160 ENDPROC
170 DEF PROCParabola
180 VDU29,94Ø;812;:GCOLØ,3
190 MOVE -3ØØ,-6ØØ
200 FOR t=-1 TO -Ø.Ø5 STEP Ø.1
210 x=-3ØØ*(t^2):y=6ØØ*t
220 angle=ATN((x+3ØØ)/-y)
230 PROCSwimmer(x,y,angle)
240 MOVE -28Ø+y,-3ØØ:DRAW
   -24Ø+y,-3ØØ
250 DRAW -26Ø+y,-28Ø:MOVE
   -24Ø+y,-3ØØ
260 DRAW -26Ø+y,-32Ø
270 NEXT t
280 MOVE -95Ø,-822
290 ENDPROC
300 DEF PROCSwimmer(x,y,angle)
310 VDU29,94Ø+x;812+y;
320 PROCRotate(Ø,3Ø,angle)
330 MOVE xt,yt
340 RESTORE 41Ø
350 FOR n=1 TO 16
360 READ a,b
370 PROCRotate(a,b,angle)
```

```
380 DRAW xt,yt: NEXT n
390 VDU29,940;812;:ENDPROC
410 DATA Ø,3Ø,–3Ø,Ø,Ø,3Ø,3Ø,Ø,Ø,3Ø,Ø,
    7Ø,3Ø,6Ø,Ø,7Ø,–3Ø,6Ø,Ø,7Ø,Ø,9Ø,1Ø,
    9Ø,1Ø,11Ø,–1Ø,11Ø,–1Ø,9Ø,Ø,9Ø
42Ø DEF PROCRotate(x,y,angle)
43Ø xt=x*COS(angle)–y*SIN(angle)
44Ø yt=x*SIN(angle)+y*COS(angle)
45Ø ENDPROC
```

```
1Ø PMODE3,1:PCLS:SCREEN1,Ø
2Ø GOSUB5Ø
3Ø GOSUB2ØØ
4Ø GOTO4Ø
5Ø COLOR3,2:LINE(Ø,38)–(255,153),
   PSET,BF
6Ø DRAW"BM123,158C4S16ND2RDNLD"
7Ø DRAW"BM123,25NRDNRD"
8Ø DRAW"BM18Ø,25RD2LU2"
9Ø RETURN
2ØØ FORT=–1TO–.Ø5 STEP .1
21Ø X=–6Ø*T*T:Y=12Ø*T
22Ø AN=ATN((X+6Ø)/–Y)
23Ø GOSUB3ØØ
24Ø DRAW"BM"+STR$(INT(13Ø+Y))+
    ",95"+"C2R2NGH"
25Ø NEXT
26Ø RETURN
3ØØ XC=187+X:YC=33–Y
31Ø XX=Ø:YY=6:GOSUB42Ø
32Ø DRAW"BM"+STR$(INT(XC+XT))+
    ","+STR$(INT(YC–YT))
33Ø RESTORE
34Ø FORN=1TO16
35Ø READ XX,YY
36Ø GOSUB42Ø
37Ø LINE–(XC+XT+.5,YC–YT+.5),
    PRESET
38Ø NEXT
39Ø RETURN
41Ø DATA Ø,6,–6,Ø,Ø,6,6,Ø,Ø,6,Ø,14,6,12,
    Ø,14,–6,12,Ø,14,Ø,18,2,18,2,22,
    –2,22,–2,18,Ø,18
42Ø XT=XX*COS(AN)–YY*SIN(AN)
43Ø YT=XX*SIN(AN)+YY*COS(AN)
44Ø RETURN
```

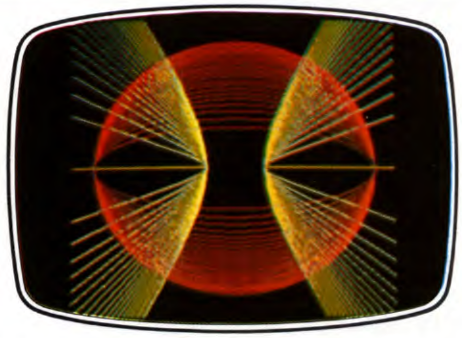
The first section of the program from Lines 5Ø to 16Ø draws the river and the banks. The next section from Lines 17Ø to 29Ø uses the equation of a parabola (Line 21Ø) to calculate the position of the swimmer. The swimmer routine at Lines 3ØØ to 4ØØ is then called, and this uses the rotate routine at Lines 42Ø to 45Ø to make sure the swimmer is drawn at the correct angle, so he is always aiming at the same spot on the bank (which in this case is the focus of the parabola). The actual shape of the swimmer is drawn from the DATA statements at Line 41Ø to 415.

CIRCLES AND POLYGONS

As far as the computer is concerned, a circle is just a many-sided polygon. The more sides the circle has, the smoother it looks. This fact is made use of in the next program which uses the equation of a circle to construct all sorts of different polygons:

```
1Ø LET ox=1ØØ: LET oy=9Ø
2Ø LET polygon=26Ø
3Ø PAPER Ø: INK 6
4Ø BORDER Ø
5Ø CLS
8Ø PRINT INK 7;AT Ø,22;"Radius of"'TAB
   22;"Circle is"'TAB 22;"82 units."
9Ø PRINT INK 4;AT 21,Ø;"Enter angle";
95 CIRCLE 1ØØ,9Ø,82
1ØØ INPUT a
1Ø5 PRINT AT 21,Ø;"□□□□□□□□
    □□□"
11Ø GOSUB polygon
12Ø PRINT AT 21,6;"Again (Y/N) ?"
13Ø LET a$=INKEY$: IF a$="y" THEN
    GOTO 2Ø
14Ø IF a$="n" THEN STOP
15Ø GOTO 13Ø
26Ø LET a=a/(18Ø/PI)
27Ø LET at=Ø
28Ø LET t=2*a
29Ø PLOT ox+82,oy
3ØØ FOR n=Ø TO 15
31Ø LET b=82*COS (t)+ox: LET c=82*SIN
    (t)+oy
32Ø LET b=(b–(PEEK 23677)): LET
    c=(c–(PEEK 23678)): DRAW b,c
33Ø BEEP Ø.Ø1,(n*5)–2Ø
34Ø LET t=t+2*a
35Ø NEXT n
36Ø RETURN
```

```
1Ø HIRES 1,6
5Ø POKE 54296,15:POKE 54278,253
6Ø COLOUR 6,1
7Ø CIRCLE 16Ø,1ØØ,7Ø,7Ø,1:
   CIRCLE 16Ø,1ØØ,6Ø,6Ø,1
75 TEXT 1Ø,18Ø,"RADIUS OF CIRCLE=4ØØ
   UNITS",1,3,11
8Ø FOR G=1 TO 2ØØØ:NEXTG:CSET(Ø)
9Ø INPUT "♥GIVE ANGLE";A
1ØØ A=A*ATN(1)/45
11Ø CSET(2):GOSUB 26Ø
13Ø TEXT 4Ø,Ø,"AGAIN (Y/N)?",1,2,2Ø
14Ø GET A$:IF A$="Y" THEN RUN
15Ø IF A$<>"N" THEN 14Ø
16Ø PRINT "♥":END
26Ø TH=2*A
27Ø N=Ø
28Ø XX=218:YY=1ØØ
```



Superimpose the conic curves for some interesting graphics

```
3ØØ LINE XX,YY,16Ø+58*COS(TH),
    1ØØ–58*SIN(TH),1
3Ø5 XX=16Ø+58*COS(TH):
    YY=1ØØ–58*SIN(TH)
31Ø TH=TH+2*A
32Ø POKE 54276,33:POKE 54273,1ØØ–N*2
33Ø N=N+1:FOR D=1 TO 1ØØ:NEXT D
34Ø IF N<15 THEN 3ØØ
35Ø POKE 54273,Ø:POKE 54276,Ø:RETURN
```

```
1Ø GRAPHIC 2:SCNCLR
5Ø POKE 36878,15
6Ø COLOR 1,6,Ø,Ø:GOSUB 7Ø:GOTO 8Ø
7Ø CIRCLE 1,512,512,29Ø,29Ø:
   CIRCLE 1,512,512,25Ø,25Ø
75 CHAR 17,Ø,"RADIUS OF CIRCLE
   □□□□=4ØØ UNITS":RETURN
8Ø FOR G=1 TO 2ØØØ:NEXTG:GRAPHIC Ø
9Ø PRINT "♥GIVE ANGLE":INPUT A
1ØØ A=A*ATN(1)/45
11Ø GRAPHIC 2:GOSUB 7Ø:GOSUB 26Ø
13Ø CHAR Ø,Ø,"AGAIN (Y/N)?"
14Ø GET A$:IF A$="Y" THEN RUN
15Ø IF A$<>"N" THEN 14Ø
16Ø GRAPHIC Ø:END
26Ø TH=2*A
27Ø N=Ø
28Ø POINT Ø,76Ø,512
3ØØ DRAW 1 TO 512+25Ø*COS(TH),
    512–25Ø*SIN(TH)
31Ø TH=TH+2*A
32Ø POKE 36876,25Ø–N*2
33Ø N=N+1:FOR D=1 TO 1ØØ:NEXT D
34Ø IF N<15 THEN 3ØØ
35Ø POKE 36876,Ø:RETURN
```

```
1Ø MODE 1
2Ø VDU19,Ø,4;Ø;
3Ø VDU24,Ø;35;1279;1Ø23;
4Ø VDU28,Ø,31,39,3Ø
5Ø VDU29,64Ø;562;
6Ø REPEAT
7Ø PROCCircle
8Ø PRINT"Radius of circle=4ØØ units"
```

```
90 INPUT"PLEASE give angle □"A
100 angle=RAD(A)
110 PROCPolygon(angle)
120 INPUT"Again(Y/N)?"Ans$
130 IF Ans$="Y" THEN CLG ELSE END
140 UNTIL FALSE
150 END
160 DEF PROCCircle
170 MOVE 400,Ø
180 FOR a=Ø TO 6.3 STEP Ø.1
190 DRAW 400*COS(a),400*SIN(a)
200 NEXT a
240 ENDPROC
250 DEF PROCPolygon(angle)
260 theta=2*angle
270 n=Ø
280 GCOLØ,2: MOVE 400,Ø
290 REPEAT
300 DRAW 400*COS(theta),400*SIN(theta)
310 theta=theta+2*angle
320 SOUND Ø,-15+n,92,1
330 n=n+1
340 UNTIL n=15
350 GCOLØ,3
360 ENDPROC
```

```
10 PMODE3,1
60 PCLS:SCREEN1,Ø
70 CIRCLE(127,95),70,4:CIRCLE(127,95),
   60,4:PAINT(127,30),4
80 FORG=1TO3000:NEXT:COLOR2
90 CLS:PRINT:INPUT"GIVE ANGLE□";A
100 A=A*ATN(1)/45
110 SCREEN1,Ø:GOSUB260
120 IF INKEY$="" THEN120
130 PRINT:PRINT:INPUT"AGAIN
   (Y/N)□";AN$
140 IF AN$="Y" THEN 60
150 IF AN$<>"N" THEN130 ELSECLS:END
260 TH=2*A
270 N=Ø
280 DRAW"BM185,95"
300 LINE-(127+58*COS(TH),95-58*
   SIN(TH)),PSET
310 TH=TH+2*A
320 PLAY"T20V"+STR$(31-N*2)+"C"
330 N=N+1
340 IF N<15 THEN300 ELSERETURN
```

The program draws a circle then asks you to INPUT the angle the first line makes to the side of the circle. This is A or a in the program, and is converted to theta or t or TH in the polygon drawing routine at Line 26Ø. The number of sides is restricted to 15 in Line 34Ø so as not to confuse the diagram with too many lines, and the sound is also based on this maximum.

If you INPUT a small angle, the polygon will be very close to a circle. With larger angles the lines trace out a star-shaped pattern.

COMPUTER ART

The next program starts off by drawing a family of hyperbolae with different eccentricities and then draws a family of ellipses on top. You could combine any of the conic curves to produce quite complex patterns.

```
10 BORDER Ø: PAPER Ø: INK 7: CLS
20 LET hyperbolae=80
30 LET ellipses=270
40 GOSUB hyperbolae
50 GOSUB ellipses
60 GOTO 60
80 LET ox=128: LET oy=87
90 FOR e=1 TO 2 STEP Ø.05
100 LET a=22: LET b=a*(SQR (e∧2-1))
102 LET h=1
104 LET f=ox+(a/COS (-1.396))
106 LET g=oy+(b*TAN (-1.396))
108 IF g<Ø THEN LET h=Ø
110 PLOT INVERSE 1; OVER 1;f,h
120 IF g>Ø THEN PLOT INK 6;f,g
130 FOR t=-80 TO 80 STEP 20
135 LET r=t/(180/PI)
140 LET x=a/COS (r): LET y=b*TAN (r)
142 LET c=oy+y: LET d=ox+x
150 IF h=Ø THEN LET d=f+g*
   (f-d)/(c-g):PLOT d,h: LET c=Ø
160 IF c>175 THEN LET d=d-
   ((d-PEEK23677)*(c-175)/
   (c-PEEK 23678)): LET c=175
170 DRAW INK 6;d-PEEK 23677,c-
   PEEK 23678: NEXT t
172 LET f=ox+(a/COS (1.75))
174 LET g=oy+b*TAN (1.75)
176 PLOT INVERSE 1; OVER 1;f,h
178 IF g<Ø THEN LET h=Ø
180 IF g>Ø THEN PLOT INK 6;f,g
190 FOR t=100 TO 260 STEP 20
195 LET r=t/(180/PI)
200 LET x=a/(COS (r)): LET y=b*TAN (r)
202 LET c=oy+y: LET d=ox+x
204 IF h=Ø THEN LET d=f+g*
   (f-d)/(c-g): PLOT d,h: LET c=Ø
206 LET h=1
210 IF c>175 THEN LET d=Ø-
   ((d-PEEK 23677)*(c-175)/
   (c-PEEK 23678)): LET c=175
220 DRAW INK 6;d-PEEK 23677,
   c-PEEK 23678
230 NEXT t: NEXT e
250 RETURN
270 FOR e=Ø.5 TO Ø.98 STEP Ø.04
280 LET a=100: LET b=a*
   (SQR (1-e∧2))
290 PLOT ox+a,oy
300 FOR t=Ø TO 360 STEP 10
305 LET r=t/(180/PI)
310 LET x=a*COS (r)
320 LET y=b*SIN (r)
330 DRAW x-(PEEK 23677)+ox,
   y-(PEEK 23678)+oy
340 NEXT t: NEXT e
360 RETURN
```

```
10 HIRES 1,6:MULTI 3,5,6:
   COLOUR Ø,Ø
20 C=ATN(1)/45
30 GOSUB 70
40 GOSUB 260
50 GOTO 50
70 FOR E=1 TO 1.50 STEP .04
100 A=13:B=A*SQR(E*E-1)
110 XX=80+INT(A/COS(-80*C)):
   YY=100-INT(B*TAN(-80*C))
130 FOR TH=-80 TO 80 STEP 20
140 X=A/COS(TH*C)
150 Y=B*TAN(TH*C)
160 LINE XX,YY,80+X,100-Y,.8+E
165 XX=80+X:YY=100-Y
170 NEXT TH
180 XX=80+INT(A/COS(100*C)):
   YY=100-INT(B*TAN(100*C))
190 FOR TH=100 TO 260 STEP 20
200 X=A/COS(TH*C)
210 Y=B*TAN(TH*C)
220 LINE XX,YY,80+X,100-Y,.8+E
225 XX=80+X:YY=100-Y
230 NEXT TH,E
250 RETURN
260 FOR E=45 TO Ø STEP -5
270 CIRCLE 80,100,35,E,3
280 NEXT E
290 RETURN
```

```
10 GRAPHIC 1:COLOR Ø,3,5,6
20 C=ATN(1)/45
30 GOSUB 70
40 GOSUB 260
50 GOTO 50
70 FOR E=1 TO 1.43 STEP .04
100 A=88:B=A*SQR(E*E-1)
110 POINT Ø,512+INT(A/COS(-80*C)),
   512-INT(B*TAN(-80*C))
130 FOR TH=-80 TO 80 STEP 20
140 X=A/COS(TH*C)
150 Y=B*TAN(TH*C)
160 DRAW .8+E TO 512+X,512-Y
170 NEXT TH
180 POINT Ø,512+INT(A/COS(100*C)),
   512-INT(B*TAN(100*C))
190 FOR TH=100 TO 260 STEP 20
200 X=A/COS(TH*C)
210 Y=B*TAN(TH*C)
220 DRAW .8+E TO 512+X,512-Y
230 NEXT TH,E
250 RETURN
260 FOR E=300 TO Ø STEP -40
```

```
270 CIRCLE 3,512,512,200,E
280 NEXT E: RETURN
```

```
10 MODE 1
20 VDU19,3,10;0;
30 PROCHyperbolae
40 PROCEllipses
50 VDU5:MOVE −650, −522
60 END
70 DEF PROCHyperbolae
80 VDU29,640;512;
90 FOR e=1 TO 2 STEP 0.05
100 a=100:b=a*(SQR(e∧2−1))
110 MOVE a/COS(RAD(−80)),b*TAN
   (RAD(−80))
120 GCOL0,2
130 FOR theta= −80 TO 80 STEP 20
140 x=a/COS(RAD(theta))
150 y=b*TAN(RAD(theta))
160 DRAW x,y
170 NEXT theta
180 MOVE a/COS(RAD(100)),b*TAN
   (RAD(100))
190 FOR theta=100 TO 260 STEP 20
200 x=a/COS(RAD(theta))
210 y=b*TAN(RAD(theta))
220 DRAW x,y: NEXT theta: NEXT e
250 ENDPROC
260 DEF PROCEllipses
270 FOR e=0.5 TO 0.98 STEP 0.02
280 a=500:b=a*(SQR(1−e∧2))
290 MOVE a,0:GCOL1,1
300 FOR theta=0 TO 360 STEP 10
310 x=a*COS(RAD(theta))
320 y=b*SIN(RAD(theta))
330 DRAW x,y: NEXT theta: NEXT e
360 ENDPROC
```

```
10 PMODE3,1:PCLS2:SCREEN1,0
20 C=ATN(1)/45
30 GOSUB70
40 GOSUB260
50 GOTO50
70 FORE=1TO1.25 STEP.02
100 A=22:B=A*SQR(E*E−1)
110 DRAW "BM"+STR$(128+INT(A/COS
   (−80*C)))+","+STR$(95−INT
   (B*TAN(−80*C)))
130 FORTH= −80TO80 STEP20
140 X=A/COS(TH*C)
150 Y=B*TAN(TH*C)
160 LINE−(128+X,95−Y),PSET
170 NEXT
180 DRAW"BM"+STR$(127+INT(A/COS
   (100*C)))+","+STR$(95−INT
   (B*TAN(100*C)))
190 FORTH=100TO260 STEP20
200 X=A/COS(TH*C)
210 Y=B*TAN(TH*C)
220 LINE−(127+X,95−Y),PSET
230 NEXTTH,E
250 RETURN
260 FOR E=1TO.1 STEP −.03
270 CIRCLE(127,95),95,3,E
280 NEXT: RETURN
```



The equation for a circle can be used for polygons as well

Both the ellipse and the hyperbola can be drawn with different eccentricities—this is the E or e in the programs. *E* can vary from Ø to 1 for an ellipse making the ellipse go from a circle to a straight line. The programs actually use *E* from .5 to .98 so the ellipses are all fairly open. For an hyperbola, *E* can vary from 1 to infinity but again, the programs restrict this range and only use E from 1 to 2. The greater E gets, the nearer the hyperbola gets to a straight line.

You can work out the eccentricity of an ellipse or hyperbola very easily. For an ellipse with equations X=A*COS T and Y=B*SIN T, then $E^2=B^2/A^2-1$, and you can see this equation in an equivalent form in Line 28Ø. (The Commodores, Dragon and Tandy use the CIRCLE command to draw the ellipses, so *E* can be entered directly into the command and doesn't need to be worked out first—see Line 27Ø.)

The hyperbola is similar. The equations are X=A/COS T and Y=B*TAN T, and in this case $E^2=1-B^2/A^2$. Again, a rearranged version of this is used in Line 1ØØ.



In a whispering gallery, sound from one focus is concentrated at another focus on the opposite side. The shape of the dome could be elliptical, parabolic or a combination of the two

WATCHING THE INTERRUPTS

If you've a few minutes on your hands, try this simple machine code routine which displays the computer's own internal 'clock' as a constantly updated digital timer

Your computer has an internal timer that runs at a constant speed, which it uses to regulate its operations. And you can use the timer, too, with a variety of BASIC instructions—such as PAUSE (on the Spectrum, or Commodore with Simons' BASIC), TIME (on the Acorns) and TIMER (on the Dragon and Tandy).

These BASIC instructions set the computer to count to a specified number in either 100ths or 50ths of a second, depending upon the speed of the machine's 'clock'. Many other operations also use the timer in a similar way—for example, if you program the computer to play music, you specify the duration of each note.

KEEPING TIME

In fact, regardless of whether or not your program specifies the length of the operations in such an obvious way, your computer is a constant clock watcher and always runs every program by the timer.

You can get the computer to keep time for you quite easily. All you need to do is to write a simple BASIC program loop which PRINTs up the time, pauses for one second, adds one second, then rePRINTs it. If you try this, you will discover that the pause needed is actually fractionally less than one second, because of the time the computer needs to perform the addition and the PRINTing operations.

Such a clock has two big disadvantages. The first is that it only keeps time while the computer is switched on. This may not be a severe problem; if you only want to know how long you have been working on something, it may actually be an advantage. But the second drawback is far more significant, and this is that as soon as you want to use the computer for something else, you will stop your clock. The reason, of course, is that you cannot run two BASIC programs at once. The answer is to use a machine code routine.

AN INTERRUPT CLOCK

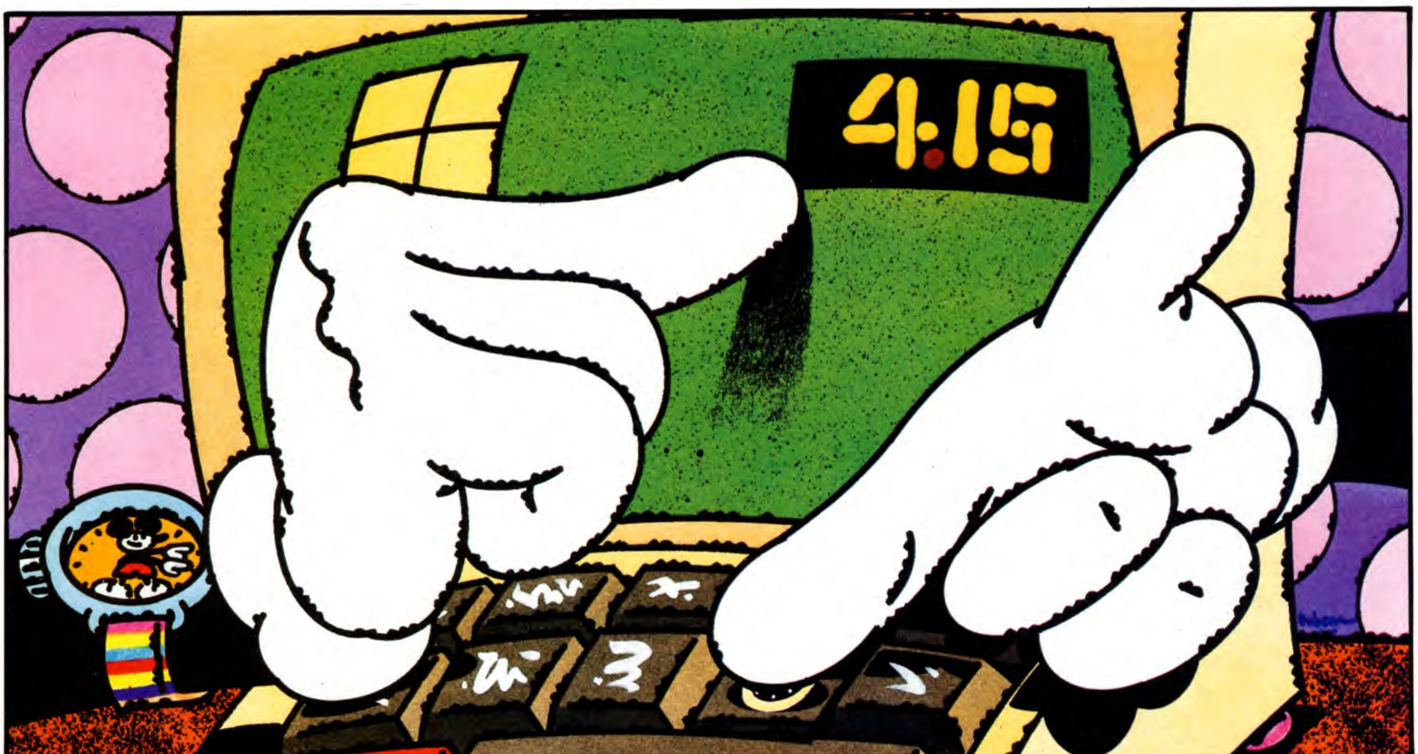
Like other machine code programs which need to run even when a BASIC program is in operation, the machine code clock which follows makes use of an interrupt-driven routine.

Spectrum and Commodore users have already seen an example of interrupts being harnessed to operate a trace program. In fact, the method for doing this is similar on all the computers.

All the time that the machine is in operation, it is constantly interrupted for a tiny fraction of a second at regular intervals. This happens even when a BASIC program is running, as the computer needs to check whether a key has been pressed. So the BASIC program is halted while the computer scans the keyboard, and then runs again until the next interrupt.

You can tack a machine code routine onto this keyboard scan in such a way that it runs in the imperceptible gaps in the BASIC program. The result is two programs which appear to run simultaneously.

Since the interrupt is itself controlled by the computer's timer, it is ideal for our purposes, as the clock can be set merely by



- THE COMPUTER AS CLOCK WATCHER
- HARNESSING THE INTERRUPTS
- THE CLOCK DISPLAY
- A SIMPLE MACHINE CODE ROUTINE
- STARTING THE CLOCK
- RESETTING THE HOURS, MINUTES AND SECONDS

counting the number of interrupts. The frequency of the interruption varies from computer to computer—the Spectrum's is every 50th of a second, while the BBC's is every 100th, for example—but the principle is the same in each case.

The programs which follow set up a simple digital clock which counts hours, minutes and seconds from the moment the clock is LOADed and turned on. You can reset the reading so that the display may be used either as a real-time clock or to count up to a set time period.

There are a few differences between the ways in which the different clocks operate. The Spectrum, Dragon and Tandy display the read-out constantly on the screen. The Electron cannot do this at all and the BBC cannot do this except in MODE 7, since the ROM routine which PRINTs up a number momentarily disables the interrupts. Since the program is written to work in any MODE, the BBC and Electron's clock can be viewed only when called up by a keypress—doing this also clears the screen.

The clocks do not keep absolutely perfect time. The routine which resets the clock on each loop does take an appreciable instant, but this is in the order of millionths of a second. So any inaccuracy is more a factor of the accuracy of your computer's timer. Even so, the clocks will keep time to within seconds per day. However, on the Spectrum, Commodores, Dragon and Tandy, SAVEing and LOADing or using BEEP, SOUND and PLAY will stop the clock for as long as the operation takes. The digital readout is constantly displayed in the top right-hand corner of the screen (except for the Acorn version) and will overwrite anything else printed there. If this is a problem, you can reorganize your screen display to miss the top line.

The following routine is suitable for either 16K or 48K machines. However, it cannot be used with Interface 1 connected, since this changes the interrupt vectors.

```
10 CLEAR 32319: LET total=0
20 FOR n=32320 TO 32554: READ a:
   POKE n,a: LET total=total+a: NEXT n
30 IF total<>24216 THEN PRINT "Error in
   data": STOP
40 RANDOMIZE USR 32320
50 DATA 33,0,0,34,120,92,34,121,92,
   62,40,237,71,237,94,201,0,64,0,0
60 DATA 62,62,237,71,237,86,201,0,
   229,213,197,245,58,91,126,60,50,91,
   126,254
70 DATA 50,32,50,175,50,91,126,58,120,
   92,60,50,120,92,254,60,32,35,175,50
80 DATA 120,92,58,121,92,60,50,121,92,
   254,60,32,20,175,50,121,92,58,122,92
90 DATA 60,50,122,92,254,13,32,5,62,1,
   50,122,92,58,122,92,38,0,111,17
100 DATA 23,64,205,234,126,58,121,92,
   38,0,111,17,26,64,205,234,126,58,120,
   92
110 DATA 38,0,111,17,29,64,205,234,
   126,17,208,61,33,29,64,205,34,127,17,
   208
120 DATA 61,33,26,64,205,34,127,62,
   120,33,24,88,119,17,25,88,1,7,0,237
130 DATA 176,205,191,2,241,193,209,
   225,251,201,237,83,80,126,1,246,255,
   205,251,126
140 DATA 1,255,255,205,251,126,201,
   175,9,60,56,252,237,66,61,198,48,229,
   205,21
150 DATA 127,33,80,126,52,42,80,126,
   205,34,127,225,201,237,75,54,92,38,0,
   111
160 DATA 41,41,41,9,235,201,6,8,26,
   119,36,19,16,250,201
```

The machine code consists of a series of DATA statements which are POKEd into memory by Line 20. As there is a large amount of DATA and it is easy to make a mistake in copying out so many numbers, Line 20 also sets up a check total—if this does not add up correctly, Line 30 stops the program with an error report, prompting you to recheck your DATA.

Line 10 moves down the start of BASIC to protect the machine code, which is called up automatically by Line 40 when you RUN this BASIC program. The clock starts at 00:00:00, but you can reset it with the following POKEs:

```
POKE 23672, (seconds)
POKE 23673, (minutes)
POKE 23674, (hours)
```

The number following the POKEs must be within the allowed range—0 to 60 for seconds and minutes, 1 to 12 for the hours. If you just want to zero the clock again, it is quicker to use:

```
RANDOMIZE USR 32320
```

which calls the machine code routine again from the start. You will also need this if you have performed a NEW, which will reset the interrupts.

```
10 S=0:FOR Z=49152 TO 49267:
   READ X:S=S+X:POKE Z,X:NEXT Z
20 IF S<>12556 THEN PRINT "ERROR IN
   DATA!":END
30 T$="000000":FOR Z=0 TO 5:
   POKE 837-Z,VAL(MID$(T$,Z+1,
   1)):NEXT Z
40 SYS 49152:PRINT"♥OK."
100 DATA 120,169,17,141,20,3,169,
   192,141,21,3,88,169,0,133,251,96
110 DATA 230,251,165,251,201,60,
   208,45,169,0,133,251,24,162,0
120 DATA 189,64,3,105,1,157,64,3,
   201,10,208,26,169,0,157,64,3
130 DATA 254,65,3,189,65,3,201,6,
   208,11,169,0,157,65,3,232,232
140 DATA 224,6,208,218,173,69,3,
   201,1,208,15,173,68,3,201,3
150 DATA 208,8,169,0,141,68,3,141,
   69,3,160,6,162,0,185,63,3,105,176
160 DATA 157,32,4,169,1,157,32,216,
   232,136,208,239,76,49,234
```

The machine code consists of a series of DATA statements which are POKEd into memory by Line 10. As there is a large amount of DATA and it is easy to make a copying error, Line 10 sets up a check total. If this does not add up correctly, Line 20 responds with an error report, prompting you to check your DATA.

The machine code is called automatically by Line 40 when you RUN this BASIC program. The clock starts at 00:00:00, but you can reset it by changing the value of T$ (set to 000000 by Line 30). The clock will be stopped if you press RUN/STOP and RESTORE. To restart it, just type:

```
SYS 49152
```

The Vic 20 program is similar to that for the Commodore 64. These are the different lines:

```
5 POKE 51,255:POKE 52,27:POKE 55,
   255:POKE 56,27:CLR
10 S=0:FOR Z=7168 TO 7283:READ
   X:S=S+X:POKE Z,X:NEXT Z
20 IF S<>12457 THEN PRINT "ERROR IN
   DATA!":END
40 SYS 7168:PRINT"♥OK."
100 DATA 120,169,17,141,20,3,169,
   28,141,21,3,88,169,0,133,251,96
160 DATA 157,14,30,169,0,157,14,
   150,232,136,208,239,76,191,234
```

You will also need to copy the Commodore 64's listing for Line 30 and the DATA statements from Line 110 to Line 150, as these are identical for both computers.

When you need to restart the Vic clock, use:

```
SYS 7168
```

```
10 MC=&900
   20 FOR T=0 TO 3 STEP 3
30 P%=MC
40 [OPT T
50 .TME BRK:BRK:BRK
60 .CLCK
70 JSR RESET
80 LDX #0
90 INC TME
100 LDA TME
110 CMP #60
120 BNE OUT
130 STX TME
140 INC TME+1
150 LDA TME+1
160 CMP #60
170 BNE OUT
180 STX TME+1
190 INC TME+2
200 LDA TME+2
210 CMP #24
220 BNE OUT
230 STX TME+2
240 .OUT
250 LDX #&FE
260 LDY #255
270 LDA #&81
280 JSR &FFF4
290 CPY #255
300 BNE O1
310 LDX #&A6
320 JSR &FFF4
330 CPY #255
340 BEQ SHOW
350 .O1
360 RTS
370 .SHOW
380 LDA #12
390 JSR &FFEE
400 LDA #31
410 JSR &FFEE
420 LDA #30
430 JSR &FFEE
440 LDA #1
450 JSR &FFEE
460 LDY #3
470 .O2
480 LDA TME-1,Y
490 JSR NUMBER
500 DEY
510 BEQ O3
520 LDA #58
530 JSR &FFEE
540 JMP O2
550 .O3
560 LDA #13
570 JMP &FFE3
580 .NUMBER
590 LDX #255
600 .N2
610 INX
620 SEC
630 SBC #10
640 BCS N2
650 ADC #58
660 PHA
670 TXA
680 ADC #47
690 CMP #48
700 BNE N3
710 LDA #32
720 .N3
730 JSR &FFEE
740 PLA
750 JMP &FFEE
760 .RESETNOS
770 ]:!P%=&FFFFFF9C:P%?4=
    &FF:P%=P%+5:[OPT T
780 .RESET
790 LDX #RESETNOS MOD 256
800 LDY #RESETNOS DIV 256
810 LDA #4
820 JMP &FFF1
830 .SETUP
840 LDA #CLCK MOD 256
850 STA &220
860 LDA #CLCK DIV 256
870 STA &221
880 LDA #14
890 LDX #5
900 JSR &FFF4
910 JMP RESET
920 ]
930 NEXT
```

When you run the above assembly language program, it will automatically be assembled by the BBC or Electron's operating system. But remember to SAVE the program before RUNning it in case of errors.

To start the clock type:

```
CALL SETUP
```

When you first assemble the program, the clock is set to 00:00:00. You can reset the time by entering:

TME?2= followed by the hours (up to 24, as this is a 24 hour clock).
TME?1= followed by the minutes.
?TME= followed by the seconds. You will have to hit [RETURN] at just the right moment to set this figure accurately.

To display the time, press [CTRL] and [DEL] simultaneously.

Tandy users should adapt the following program by altering the two numbers printed in bold in Line 140. Change **157** to 137, and change **61** to 76.

This routine is not suitable for use when a disk drive is connected.

```
10 CLEAR 200,32599
20 FOR J=32600 TO 32679
30 READ N
40 POKE J,N
50 NEXT
100 DATA 204,0,0,253,127,252,253,127,
    254,48,140,4,191,1,13,57
110 DATA 206,127,164,142,128,0,166,
    130,76,161,192,38,9,111,132
120 DATA 140,127,252,38,242,134,1,
    167,132,206,4,32,142,127,255
130 DATA 79,230,130,192,10,45,3,76,
    32,249,195,47,58,237,195,17
140 DATA 131,4,25,47,6,134,58,167,
    194,32,229,126,157,61,50,60,60,13
```

The machine code consists of a series of DATA statements which are POKEd into memory by Line 40. Any errors in your DATA will probably cause the machine to crash, so SAVE the routine before RUNning it and check all the numbers very carefully.

RUN the program to enter the machine code. To start the clock, type:

```
EXEC32600
```

This sets the clock to 00:00:00. You can reset it with the following POKEs:

```
POKE 32764, (hours)
POKE 32765, (minutes)
POKE 32766, (seconds)
```

Remember to keep each of these values within the permitted range.

MARK MY WORDS

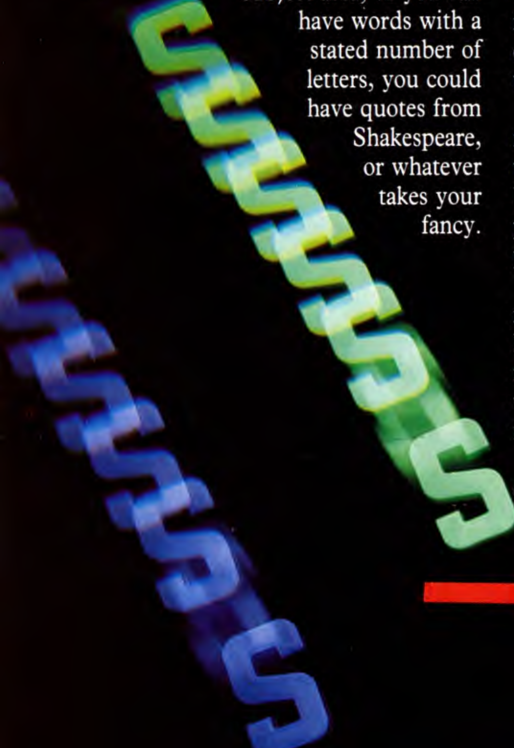
- A GAME FOR TWO PLAYERS
- SETTING UP THE SCREEN
- THE RULES OF THE GAME
- LETTER VALUES
- STRATEGY

Put in a good word for educational computer games. INPUT's word game is suitable for all ages, can be made as hard, or as easy as you wish, and is incredibly addictive

Computer games do not have to be purely recreational like arcade games, or some of the simulations available at home, they can be educational, too.

'Hangman' is one well known game which can be converted to run on a computer. The game can help people with spelling, general knowledge, general grasp of English and so on. Choose a subject like Chemical Engineering, and you'll soon pick up some of the buzzwords.

INPUT's word game comes from the same stable, being a game for two players, involving guessing words or phrases. The game is more interesting, and more fun to play than Hangman, and is just as educational. You can play it somewhat like Hangman, with a stated subject area, or you can have words with a stated number of letters, you could have quotes from Shakespeare, or whatever takes your fancy.



THE GAME

First enter the names of the two players. You then have the option of choosing the number of words in the phrases that each person enters. One interesting facet of the game is that the longer phrases are sometimes the easiest of all to guess because there are more clues—try it and see.

Once you have picked the number of words, you have to choose the number of turns that will constitute the game.

Now the first player has to dream up a phrase and enter it. The opponent doesn't have to be locked screaming in the nearest large cupboard while it's being entered, because the letters will not appear as they are typed in. But if you have a cooperative opponent, you can take the option for the letters to appear on screen as they are typed. Having the letters on screen alleviates the problem of mistyping the phrase, and the ensuing arguments when it appears.

There should only be a single space between each word in the phrase. The maximum length for any phrase is 64 characters on the Spectrum, Dragon and Tandy, 77 on the Commodore 64 and 80 on the Acorns.

Once the phrase is complete, the enter key is pressed and the main screen appears. At the top are the scores for both players. At the beginning of the game each player has 200 points, and the total may go up or down as play progresses.

Under the scores is a table of letter values, more common letters having high values, and less common letters having lower values. The mystery phrase is shown as a row of asterisks, with, in the case of the Acorn machines, a flashing underline cursor.

At the bottom of the screen display are a set of instructions, and space for entering your commands and guesses.

STRATEGY

There are three options given to the guesser: buying letters, guessing a letter at a specific position, or guessing the whole phrase.

In the earliest stages of guessing, a good choice is to buy a space—make sure that the phrase contains more than one word, though! How to proceed now is up to you. Vowels are expensive, but have a very high probability of occurrence; the cheaper letters are risky because of their rarity. The words are easier to guess once you've found some consonants—a liberal splattering of vowels is not always too helpful.

As the phrase takes shape, you will probably find that you are able to guess a letter at a specific position. For example, you may have a word that looks like this: T*E. A central H is a fairly safe guess. It's now that you can score points. A correctly chosen letter will add its value to your score, while if you guess wrongly, the loss is only half the letter's value. Press XX to select the guess option, and insert your guess by using a cursor as prompted.

With several letters in place, you may get a flash of inspiration and want to guess at the whole phrase. To do this, type ZZ, and you can enter the whole phrase. If it's correct, the score for the whole phrase—the remaining letters only, of course—is worked out and added to the player's score. If the guess is wrong, then 50 points are subtracted instead. Too many wild guesses will soon erode your score.

Now type in the first section of Wordgame. The lines set up the screen ready for play to commence. If you RUN the program, you will see this working but you won't get too far with the game because the remainder of the program (which covers the various choices) is covered in the next part.

Don't forget to SAVE the program.

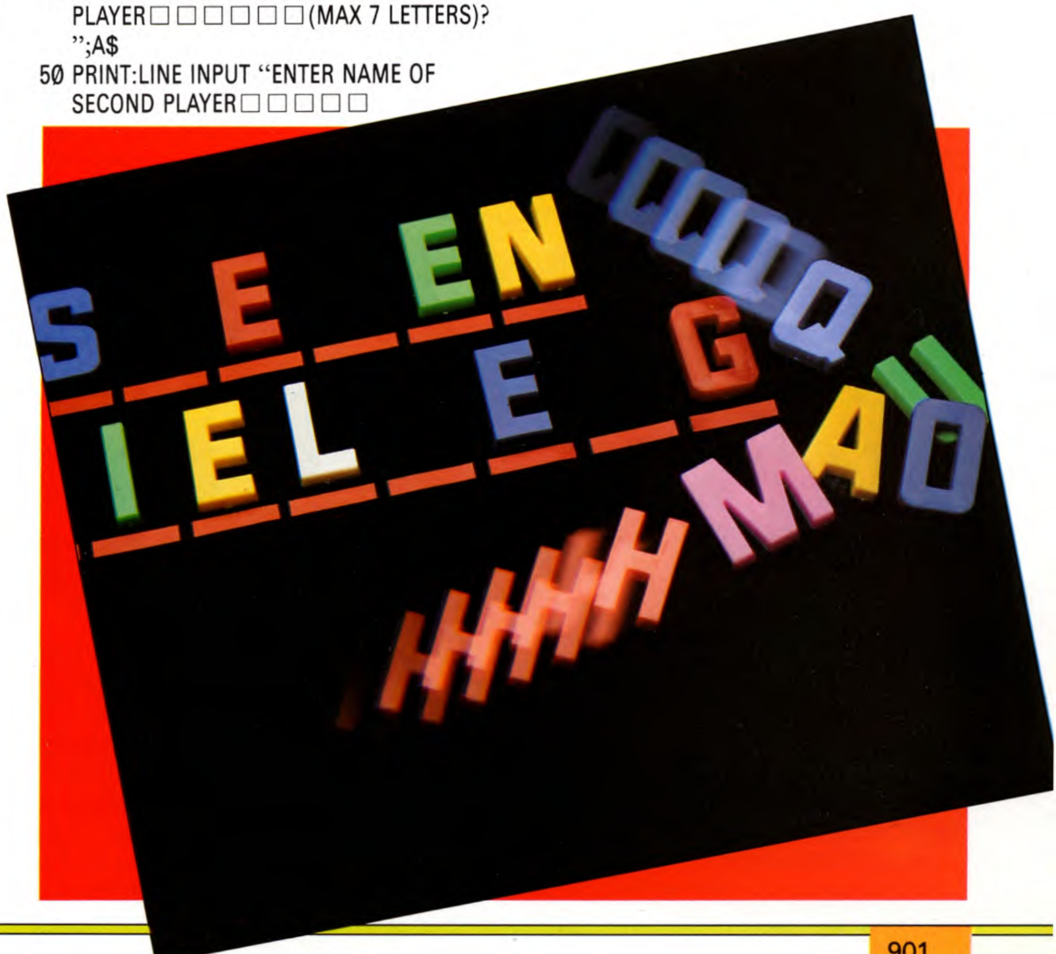


```
10 LET r$=“WORD”: LET w=14: LET d=Ø:
   LET f=1: LET g$=“”: LET g=Ø: LET
   k=Ø: LET q$=“”: LET ta=2ØØ: LET
   tb=2ØØ: LET tc=Ø: LET b=Ø: POKE
   236Ø9,5Ø: POKE 23658,8: LET i$=“”: LET
   j$=“”: LET z$=“”: LET c$=“”
20 FOR n=Ø TO 7: READ y: POKE USR
   “a”+n,y: NEXT n
30 DATA 255,129,129,129,129,129,
   129,255
40 INPUT “ENTER NAME OF FIRST
   PLAYER□□□□□□□(UP TO 7
   LETTERS)”, LINE a$
50 INPUT “ENTER NAME OF SECOND
   PLAYER□□□□□(UP TO 7 LETTERS)”,
   LINE b$
60 IF LEN a$>7 OR LEN b$>7 THEN GOTO
   4Ø
70 CLS : INPUT “ENTER NUMBER OF WORDS
   IN PHRASE (1 TO 9)”, LINE c$
80 IF LEN c$<>1 THEN GOTO 7Ø
90 IF CODE c$<49 OR CODE c$>57 THEN
   GOTO 7Ø
100 LET c=VAL c$
110 INPUT “ENTER NUMBER OF TURNS (1
   TO 9)”, LINE t$
120 IF LEN t$<>1 THEN GOTO 11Ø
130 IF CODE t$<49 OR CODE t$>57 THEN
   GOTO 11Ø
140 LET t=VAL t$
150 IF c>1 THEN LET j$=“S”: LET
   i$=“WITH A SINGLE
   SPACE□□□□□BETWEEN EACH”: LET
   r$=“PHRASE”
160 PRINT a$;“, IT IS YOUR
   TURN.”’“PLEASE ENTER YOUR PHRASE
   OF□□□□□”;c;“□WORD”;
   j$;“.□THE LETTERS YOU ENTER”’“WILL
   NORMALLY BE INVISIBLE, BUT IF YOU
   WISH TO SEE THEM PRESS Ø.OTHERWISE,
   PRESS 1 TO CONTINUE.”
170 LET k$=INKEY$: IF k$=“” THEN GOTO
   17Ø
190 IF k$=“Ø” THEN POKE 23624,56:INPUT
   LINE s$: CLS : GOTO 22Ø
200 IF k$=“1” THEN POKE 23624,63: INPUT
   LINE s$: CLS : POKE 23624,56: GOTO 22Ø
210 GOTO 17Ø
220 LET l=LEN s$
230 IF l=Ø THEN PRINT “ILLEGAL ENTRY.
   PLEASE CHECK AND RE-ENTER”: PAUSE
   1ØØ: CLS : GOTO 16Ø
240 IF l>64 THEN PRINT “ENTRY TOO
   LONG. PLEASE CHECK AND RE-ENTER”:
   PAUSE 1ØØ: CLS : GOTO 16Ø
250 FOR n=1 TO l: IF s$(n)=CHR$ 32
   THEN LET d=d+1: GOTO 27Ø
260 IF CODE s$(n)<65 OR CODE s$(n)>9Ø
   THEN PRINT “YOU HAVE ENTERED AN
   ILLEGAL□□□□□LETTER. CHECK
   AND RE-ENTER”: PAUSE 1ØØ: CLS : LET
   d=Ø: GOTO 16Ø
270 IF c=1 AND d=1 THEN PRINT “YOU
   CANNOT HAVE SPACES IN
   A□□□□□SINGLE WORD. CHECK
   AND RE-ENTER”: PAUSE 1ØØ: CLS : LET
   d=Ø: GOTO 16Ø
280 NEXT n
290 IF d<>c-1 THEN PRINT “YOU ARE
   SUPPOSED TO BE ENTERING□”;c;“□
   WORDS□”;i$;“. CHECK AND RE-ENTER”:
   PAUSE 1ØØ: CLS : LET d=Ø: GOTO 16Ø
300 LET z$=“”
310 FOR n=1 TO l: LET z$=z$+“*”:
   NEXT n
320 PRINT INK 1; AT Ø,Ø;a$;“’S SCORE ”:
   PRINT INK 1;AT Ø,16;b$;“’S SCORE□□”:
   PRINT PAPER 2, INK 6;AT 1,6;ta;TAB
   22;tb;TAB 31;“□”
330 PRINT AT 3,7;“CHARACTER VALUES”
340 FOR n=Ø TO 26: READ g$: LET
   q$=q$+g$: NEXT n: PRINT q$:
   RESTORE 9ØØ
350 PRINT INK 1;AT 12,Ø;“THE□”;r$;
   “□”;b$;“□HAS TO
   GUESS”’“CONTAINS□”□“□CHARACTER
   S”: PRINT PAPER 2; INK 6;z$
360 INPUT “DO YOU WANT TO BUY A
   CHARACTER AT□□THE POINTS PRICE
   SHOWN? ENTER□□□YOUR
   CHARACTER CHOICE. OTHERWISE ENTER
   XX TO GUESS A CHARACTER OR ZZ TO
   GUESS THE WHOLE PHRASE.”, LINE d$
1000 DATA “A-2Ø□□□□”,
   “B-1Ø□□□□”,“C-1Ø□□□□”,
   “D-12□□□□”,“E-2Ø□□□□”,
   “F-Ø8□□□□”,“G-12□□□□”,
   “H-Ø8□□□□”
1010 DATA “I-2Ø□□□□”,
   “J-Ø4□□□□”,“K-Ø6□□□□”,
   “L-1Ø□□□□”,“M-1Ø□□□□”,
   “N-1Ø□□□□”,“O-2Ø□□□□”,
   “P-1Ø□□□□”,“Q-Ø2□□□□”,
   “R-12□□□□”,“S-12□□□□”
1020 DATA “T-12□□□□”,
   “U-2Ø□□□□”,“V-Ø8□□□□”,
   “W-Ø8□□□□”,“X-Ø4□□□□”,
   “Y-Ø8□□□□”,“Z-Ø2□□□□”,
   “<graphics a>-2Ø□□□□”
```

```
5 POKE 5328Ø,1:POKE 53281,1:PRINT
   “♥▪”;CHR$(8)
6 SP$=“▪”:FOR Z=1 TO 39:SP$=
   “□”+SP$:NEXT Z
10 R$=“WORD”:W=14:F=1:TA=2ØØ:
   TB=2ØØ
16 QD$=“▪”:FOR Z=1 TO 23:
```

```
   QD$=QD$+"■":NEXT Z
40 A$="JACK":PRINT "♥ENTER NAME OF
   PLAYER 1":INPUT A$:A$=LEFT$(A$,11)
50 B$="JILL":PRINT "♥ENTER NAME OF
   PLAYER 2":INPUT B$:B$=LEFT$(B$,11)
70 PRINT "♥ENTER DIFFICULTY LEVEL
   (NUMBER OF WORDS IN PHRASE 1-9)"
90 GET C$:C=VAL(C$):IF C<1 OR C>9
   THEN 90
110 PRINT "♥ENTER NUMBER OF TURNS
   (1-9)"
130 GET T$:T=VAL(T$):IF T<1 OR T>9
   THEN 130
150 IF C>1 THEN J$="S":I$=
   "WITH A SINGLE SPACE BETWEEN
   EACH":R$="PHRASE"
155 PRINT "♥"
160 PRINT A$;", IT IS YOUR
   TURN":PRINT"■PLEASE ENTER YOUR
   PHRASE OF";C;
162 PRINT "WORD";J$:PRINT "IF YOU WISH
   TO SEE THE LETTERS YOU"
165 PRINT "ENTER THEN PRESS 'Ø', ELSE
   PRESS '1' TO CONTINUE . . . ■"
170 GET K$:IF K$="" THEN 170
190 IF K$="1" THEN PRINT "?■": INPUT
   S$:PRINT "■":GOTO 220
200 IF K$="Ø" THEN INPUT S$:GOTO 220
210 GOTO 170
220 L=LEN(S$):PRINT
230 IF L=Ø THEN PRINT "ILLEGAL ENTRY—
   REDO":GOSUB 950:GOTO 155
240 IF L>64 THEN PRINT "ENTRY TOO
   LONG—REDO":GOSUB 950:GOTO 155
250 FOR N=1 TO L:IF MID$(S$,N,1)
   =CHR$(32) THEN D=D+1:GOTO 270
260 IF MID$(S$,N,1)<"A" OR
   MID$(S$,N,1)>"Z" THEN 265
263 GOTO 270
265 PRINT "ILLEGAL CHARACTER—
   REDO":GOSUB 950:D=Ø:GOTO 155
270 IF C=1 AND D=1 THEN 275
273 GOTO 280
275 PRINT"SPACES ARE NOT ALLOWED
   IN A SINGLE WORD!—REDO":
   GOSUB95Ø:D=Ø:GOTO155
280 NEXT N
290 IF D<>C-1 THEN 295
293 GOTO 300
295 PRINT"YOU ARE MEANT TO ENTER
   □";C;"□WORDS□";I$;" —REDO":
   GOSUB95Ø:D=Ø:GOTO155
300 Z$=""
310 FOR N=1 TO L:Z$=Z$+"*":
   NEXT N
320 PRINT "♥■"A$;"'S SCORE";
   TAB(2Ø);B$;"'S SCORE":PRINTSP$;
   "■"TA;TAB(2Ø);TB
330 PRINT "■■□□□□□□□□
   □□□□□CHARACTER VALUES
   □□□□□□□□□□□□□□
   ■■";
340 FOR N=Ø TO 26:READ G$:Q$=
   Q$+G$+"■":NEXT N:PRINT
   Q$:RESTORE:GOSUB 2000
350 PRINT LEFT$(QD$,1Ø)"■
   THE□";R$"□CONTAINS";L;
   "LETTERS■":PRINTZ$
360 PRINT LEFT$(QD$,17)"■■■■
   XX=GUESS LETTER□,□ZZ=GUESS
   PHRASE"
370 PRINTTAB(6)"A—SPACE=BUY
   THAT CHARACTER■":D$="":
   PRINT"■■"SP$"■□■":
   INPUT D$
900 DATA A-2Ø,B-1Ø,C-1Ø,D-12,
   E-2Ø,F-Ø8,G-12,H-Ø8
910 DATA I-2Ø,J-Ø4,K-Ø6,L-1Ø,
   M-1Ø,N-1Ø,O-2Ø,P-1Ø,Q-Ø2,
   R-12,S-12
920 DATA T-12,U-2Ø,V-Ø8,
   W-Ø8,X-Ø4,Y-Ø8,Z-Ø2,
   "▒-2Ø"
```

```
5 CLEAR 1ØØØ
10 R$="WORD":W=14:F=1:
   TA=2ØØ:TB=2ØØ
15 P1=PEEK(359):P2=PEEK(36Ø):
   P3=PEEK(361)
40 CLS:LINE INPUT "ENTER NAME OF FIRST
   PLAYER□□□□□□(MAX 7 LETTERS)?
   ";A$
50 PRINT:LINE INPUT "ENTER NAME OF
   SECOND PLAYER□□□□□
   (MAX 7 LETTERS)? ";B$
60 IF LEN(A$)>7 OR LEN(B$)>7 THEN 4Ø
70 CLS:LINE INPUT "ENTER DIFFICULTY
   LEVEL (NUMBER□□OF WORDS IN
   PHRASE 1-9)?□";C$
80 IF LEN(C$)<>1 THEN 7Ø
90 IF C$<"1" OR C$>"9" THEN 7Ø
100 C=VAL(C$)
110 PRINT:LINE INPUT "ENTER NUMBER OF
   TURNS (1-9)?□";T$
120 IF LEN(T$)<>1 THEN 11Ø
130 IF T$<"1" OR T$>"9" THEN 11Ø
140 T=VAL(T$)
150 IF C>1 THEN J$="S":I$="WITH A
   SINGLE SPACE BETWEEN EACH":
   R$="PHRASE"
155 CLS
160 PRINT A$;",□IT IS YOUR
   TURN":PRINT:PRINT"PLEASE ENTER
   YOUR PHRASE OF□";C,"□WORD";J$
165 PRINT:PRINT"IF YOU WISH TO SEE THE
   LETTERS□□YOU ENTER THEN PRESS
   'Ø', ELSE PRESS '1' TO CONTINUE
   . . .":PRINT
170 K$=INKEY$:IF K$="" THEN 17Ø
190 IF K$="1" THEN PRINT"?□";:
   POKE 359,&H86:POKE 36Ø,32:POKE
   361,57:LINE INPUT S$:POKE 359,
   P1:POKE 36Ø,P2:POKE 361,P3:GOTO 22Ø
```



```
200 IF K$="Ø" THEN LINE INPUT
    "?□";S$:GOTO 220
210 GOTO 170
220 L=LEN(S$):PRINT
230 IF L=Ø THEN PRINT"ILLEGAL ENTRY—
    REDO":GOSUB 950:CLS:GOTO160
240 IF L>64 THEN PRINT"ENTRY TOO
    LONG—REDO":GOSUB 950:CLS:GOTO
    160
250 FOR N=1 TO L:IF MID$(S$,N,1)
    =CHR$(32) THEN D=D+1:GOTO 270
260 IF MID$(S$,N,1)<"A" OR
    MID$(S$,N,1)>"Z" THEN PRINT
    "ILLEGAL CHARACTER—REDO":GOSUB
    950:CLS:D=Ø:GOTO 160
270 IF C=1 AND D=1 THEN PRINT
    "SPACES ARE NOT ALLOWED IN
    A□□□□□SINGLE WORD!—REDO":
    GOSUB 950:CLS:D=Ø:GOTO160
280 NEXT N
290 IF D<>C-1 THEN PRINT"YOU
    ARE MEANT TO ENTER";C;
    "WORDS□□";I$;"—REDO":
    GOSUB950:CLS:D=Ø:GOTO160
300 Z$=""
310 FOR N=1 TO L:Z$=Z$+"*":NEXT N
320 CLS:PRINT A$;"'S SCORE",
    B$;"'S SCORE":PRINT@38,TA;
    TAB(22);TB;"□□"
330 PRINT@7Ø,"character values"
340 FOR N=Ø TO 26:READ G$:Q$=
    Q$+G$:NEXTN:PRINTQ$:RESTORE
350 PRINT@320,"THE□";R$;
    "□CONTAINS";L;"LETTERS":
    PRINTZ$
360 PRINT@416,"";:LINE INPUT
    "XX=GUESS LETTER□ZZ=GUESS
    PHRASE A-Z=BUY THAT CHARACTER ?
    ";D$
900 DATA "A-20□□□□",
    "B-10□□□□","C-10□□□□",
    "D-12□□□□","E-20□□□□",
    "F-Ø8□□□□","G-12□□□□",
    "H-10□□□□"
910 DATA "I-20□□□□",
    "J-Ø4□□□□","K-Ø6□□□□",
    "L-10□□□□","M-10□□□□",
    "N-10□□□□","O-20□□□□",
    "P-10□□□□","Q-Ø2□□□□",
    "R-12□□□□","S-12□□□□"
920 DATA "T-12□□□□",
    "U-20□□□□","V-Ø8□□□□",
    "W-Ø8□□□□","X-Ø4□□□□",
    "Y-Ø8□□□□","Z-Ø2□□□□",
    "s-20□□□"
```

The programs for each of the machines are very broadly similar, as there are no graphics which require the use of the machines' special facilities to display them.

Initializing the programs is a little different in some cases. In the Commodore program, Line 5 sets up the screen colours, whilst in the case of the Dragon Line 5 CLEARs sufficient string space for the game.

Line 1Ø initializes all the strings and variables needed in the game. The line in the Dragon and the Commodore program is noticeably shorter than in the other programs because there's no need to initialize variables to zero, or to initialize null strings. The PEEKs in Line 15 of the Dragon program are used later in the program to stop PRINTed material appearing on the screen, and they work by intercepting the machine's PRINT routine. Lines 2Ø and 3Ø in the Spectrum program set up the space UDG used in the letter table.

Lines 4Ø to 7Ø are the prompts right at the start of the game. Lines 4Ø and 5Ø are the names of the first and second players, and Line 6Ø checks that they are not too long for the screen space allotted. The number of words in the phrase are input in Line 7Ø.

The following routine, from Lines 8Ø to 1ØØ is a series of validity checks, making sure that the number of words in the phrase is within the limits of the program.

The number of words in the phrase is called C$ (or c$), and Line 8Ø checks that the input is only one character long. Line 9Ø checks that the input is between 1 and 9—between them the two lines check that the input is between 1 and 9 and is a whole number. Line 1ØØ converts the string into a numeric variable.

Lines 11Ø to 14Ø are related to the number of turns chosen. Line 11Ø is the prompt and calls the number of turns, T$ or t$. Lines 12Ø and 13Ø are similar validity checks to before, whilst Line 14Ø converts the string to a numeric variable.

Line 15Ø checks that the number of words in the phrase is greater than one, and tells the players that there needs to be a single space between each word. R$ or r$ is set to equal "PHRASE"—used later on in the prompts.

The program now enters the input routine. This extends from Line 16Ø to 22Ø, and gives instructions to the player whose turn it is to type in the mystery phrase. Selecting Ø will make the phrase appear on screen as it is typed in—otherwise it is invisible. The phrase is called S$ or s$.

Lines 23Ø to 29Ø are validity checks. If the length of the phrase is less than one character—if the player has just pressed RETURN or ENTER—Line 23Ø announces that it is an illegal entry. Line 24Ø checks if the entry is the right length, and Line 25Ø checks for the number of spaces (which must be one less than the number of words given by C or c). Lines 26Ø and 27Ø checks for illegal entries along with Line 29Ø.

Lines 3ØØ and 31Ø set up Z$ or z$—a dummy string consisting of asterisks but equal in length to S$ or s$.

The final routine, from Lines 32Ø to 36Ø sets up the remainder of the main screen, READing a table of letter values from the DATA at the end of the program, PRINTing up the two players scores and displaying the dummy string containing the asterisks. Line 36Ø is a prompt to the guesser.

```
10 *FX4,1
20 ON ERROR GOTO 900
30 MODE6:DIMA$(2),S(2),V(27)
40 PROCSETUP
50 PROCWORD
60 FOR TQ=1 TO NG
70 FOR TP=1 TO 2
80 PROCSCREEN
160 DEF PROCSCREEN
170 CLS:RESTORE
180 PRINT:PRINTA$(1);"'S SCORE"
    TAB(2Ø)A$(2);"'S SCORE"
190 PRINT:PRINT:FOR T=1 TO 27:READ
    A$:V(T)=EVAL(MID$(A$,3-(T=27)
    *2,2)):PRINTA$SPC6;:NEXT:
    PRINT:PRINT
200 PRINT"THE WORD□";A$(3-P);
    "□HAS TO GUESS""CONTAINS□";
    L;"□CHARACTERS"
210 PRINTTAB(Ø,16)Z$
220 PRINTTAB(Ø,22)"ENTER A LETTER,
    MOVE THE CURSOR, PRESS□□
    'CTRL B' TO BUY OR 'CTRL G' TO GUESS
    THE COMPLETE PHRASE";
230 ENDPROC
470 DEF PROCSETUP
480 INPUT"ENTER 1ST PLAYERS NAME",
    A$(1):A$(1)=LEFT$(A$(1),1Ø)
490 PRINT:INPUT"ENTER 2ND PLAYERS
    NAME",A$(2):A$(2)=LEFT$
    (A$(2),1Ø)
500 PRINT
510 INPUT"HOW MANY WORDS IN EACH
    PHRASE (1 TO 9)",NW
520 NW=INT(NW):IF NW<1 OR NW>9
    THEN 510
530 INPUT"HOW MANY TURNS (1 TO 9)",NG
540 NG=INT(NG):IF NG<1 OR NG>9
    THEN 530
550 S(1)=2ØØ:S(2)=2ØØ:P=1
560 ENDPROC
570 DEF PROCWORD
580 X=1:G=Ø
590 CLS:PRINTTAB(Ø,2)"O.K.□";A$(P):
    PRINT"ENTER YOUR□";NW;
    "□WORD PHRASE"
600 PRINT:PRINT"IF YOU WISH TO SEE THE
    LETTERS PRESS 'Ø'ELSE PRESS ANY KEY
```




```
     AND CONTINUE TO TYPE"
610 K=GET-48
620 IF K<>Ø THEN COLOURØ
630 INPUT""Y$:Z$=STRING$(LEN(Y$),
     "*"):L=LENY$:B$=""
640 COLOUR3
650 IF L=Ø OR L>8Ø THEN PRINT
     "RE-ENTER THE PHRASE IT IS NOT
     THE CORRECT LENGTH":
     GOTO 62Ø
660 TK=Ø:FOR T=1 TO L:TK=TK-
     (MID$(Y$,T,1)="□"):IF MID$
     (Y$,T,1)<>"□" AND (MID$
     (Y$,T,1)<"A" OR MID$(Y$,T,1)
     >"Z") THEN TK=-1:T=L
670 NEXT
680 IF TK=-1 THEN PRINT"YOU HAVE
     ENTERED AN ILLEGAL LETTER
     □□□□□□RETYPE IT":
     GOTO 62Ø
690 IF TK<>NW-1 THEN PRINT"YOU
     HAVE TO ENTER□";NW;"□WORDS WITH
     SINGLE□□□SPACES SEPARATING
     THEM":GOTO 62Ø
7ØØ ENDPROC
9ØØ *FX4,Ø
91Ø PRINT:REPORT:PRINT"□AT LINE□";
     ERL:END
97Ø DATA A-2Ø,B-1Ø,C-1Ø,D-12,
     E-2Ø,F-Ø8,G-12,H-Ø8
98Ø DATA I-2Ø,J-Ø4,K-Ø6,L-1Ø,
     M-1Ø,N-1Ø,O-2Ø,P-1Ø,Q-Ø2,
     R-12,S-12
99Ø DATA T-12,U-2Ø,V-Ø8,W-Ø8,
     X-Ø4,Y-Ø8,Z-Ø2,[□]-2Ø
```

In the Acorn program, Line 1Ø makes the cursor keys generate ASCII codes. Line 2Ø prints out any error messages by jumping to Lines 9ØØ and 91Ø. Line 3Ø sets MODE 6 and DIMensions three arrays.

PROCSETUP can be found starting at Line 47Ø. It allows the two players' names to be entered, along with the number of words that will be found in each phrase, and the number of turns.

Next, PROCWORD is called. PROCWORD starts at Line 57Ø, and allows the first player to enter the phrase that the second player has to guess. The phrase is checked for correct length, having allowable letters, and single spaces.

Lines 6Ø and 7Ø are the start of FOR ... NEXT loops for the number of goes, and the number of players, and will get their NEXTs next time.

Finally, PROCSCREEN looks after the screen display. Starting at Line 16Ø, the players' names and scores are displayed, along with the letter values assigned and some instructions.

CLIFFHANGER: PROGRAM A COMPLETE ARCADE GAME

Can you learn assembly language before the goats eat Willie's picnic? Start to program a fully playable arcade-style machine code game— part two follows next issue

There are many serious business applications of machine code programming. But all of the important principles can be outlined in games programming—and learning how to program in dry machine code is much more fun when you apply it to writing games.

So *INPUT* is giving you a complete machine-code game, specially constructed to cover the main programming faculties on the 48K Spectrum, Commodore 64, BBCB and Dragon. This will show you how a typical game is constructed and how the various programming elements are combined to produce interesting graphics, smooth action and exciting effects.

THE GAME

INPUT's game is called Cliffhanger. It is a running and jumping game of the Donkey Kong/Hunchback variety and has four screens which get progressively more difficult. The main character is Willie, who has been out having a picnic on the cliffs at the seaside. He has taken a short walk to build up his appetite. But when he returns he finds that some goats have spread his picnic all over a rocky embankment. Willie has to climb to the top of the cliff to reclaim his lost possessions. This is made all the more urgent by the fact that the tide is rising and Willie is in danger of being drowned if he does not get to the top of the cliff in time.

In the first screen he is hampered by falling rocks. These come rolling down the slope and he has to jump over them. One slip could mean sudden death. You control his running and jumping with the N and M keys. If he's hit by a boulder, Willie is buried immediately. Luckily he has five lives.

When he makes it to the top of the slope and reclaims the first item of his missing picnic he is returned to the bottom of the slope again and moves onto the second screen. This time when he tries to scale the cliff, he has to jump over pot holes. If he falls down one he gets buried again.

When Willie has reached the top on the second screen, he moves onto the third. Again he has to scale the slope, avoiding potholes. But this time they are inhabited by vicious snakes which try and bite him as he leaps over the hole. And on the fourth screen, Willie has to contend with snakes, potholes and boulders.

On each screen Willie must keep ahead of the rising tide. He gets points for climbing the slope and a bonus for collecting four of his picnic items without losing a life.

As well as being an education in program writing, Cliffhanger is fun to play. It will be published as a serial in consecutive parts in *INPUT*. How each part works and how it fits into the overall structure of the game will be fully explained. Examples of how certain routines can be used in different



- HOW THE GAME IS STRUCTURED
- BASIC **POKE**R PROGRAMS
- CONSTRUCTING ASCII DATA TABLES
- OUTPUTTING TO THE SCREEN
- ROM PRINT ROUTINES
- PAUSING
- DEBUGGING LOOPS

applications will also be given.

And at the end of all that, you will have a game that is as good as many available commercially.



THE GAME'S STRUCTURE

The background and the moving characters are all made using UDGs, except on the Commodore 64 which uses sprites. The general background is generated using loops to PRINT them on the screen.

The potholes and snakes are superimposed on the first screen. That way, most of the background does not have to be redefined to make the second and third screens.

The main part of the program comprises an executive routine which controls the timing and priority of events. The events themselves are added as subroutines. The executive is driven by interrupts (see page 478).

Except on the Commodore 64, where sprites are used, the movement of the boulders and the man are made in half character jumps. This is accomplished using two sets of characters and gives acceptably smooth action without making the program too complex and slowing it down. In any game of this sort, speed is important.

The first thing that has to be done in any game is to print the title page on the screen. Although the print routine is in machine code, there is little point in supplying the words to be printed in machine code. Instead the words you want printed on the screen are typed in as part of the following BASIC program, which then POKEs them into the protected part of memory. You must of course key in CLEAR 57434 first.

```
10 LET X=57435
20 READ A$
30 FOR N=1 TO 45
40 POKE X,CODE A$(N)
50 LET X=X+1
60 NEXT N
70 DATA"CLIFFHANGERCREATED BY
   A.DOEWRITTEN BY P.CLARK"
```

This program POKEs the title-page data into a data table. The resulting portion of memory should then be SAVEd to tape. Then LOAD your assembler and key in the main machine code routine which follows:

```
        org 58Ø35
ti      call cl
        ld a,2
        out (254),a
        ld a,16
        ld (23624),a
        ld ix,57435
        ld b,5
        ld a,7Ø
        ld hl,134
        call me
        ld b,6
        ld hl,2Ø4
        call me
        ld b,16
        ld a,7
        ld hl,61Ø
        call me
        ld b,18
        ld ld hl,674
        call me
        ld b,2
ldp     ld hl,65ØØØ
        ld de,Ø
ldq     dec hl
        push hl
        sbc hl,de
        pop hl
        jr nz,ldq
        djnz ldp
        ret
        org 58192
cl      *
        org 58155
me      *
```

Now key in the following service routines and assemble in the same way.

SAVE the source code using the SAVE option on the assembler. Then assemble the code, NEW and LOAD your machine code monitor. Then you should SAVE the object code onto tape as well.

```
        org 58146
ktt     ld a,253
        in a,254
        bit 1,a
        jr nz,ktt
        ret
me      push bc
        push af
        ld a,(ix+Ø)
        call asc
        pop af
        call print
        inc hl
        inc ix
        pop bc
        djnz me
        ret
asc     push hl
        ld hl,156Ø8
        ld de,8
        ld b,31
        sub b
ash     add hl,de
        dec a
        jr nz,ash
        push hl
        pop bc
        pop hl
        ret
cl      ld ix,16384
        ld hl,6912
        ld a,Ø
clp     ld (ix+Ø),a
        inc ix
        dec hl
        push hl
        ld de,Ø
        sbc hl,de
        pop hl
        jr nz,clp
        ret
print   push af
        push hl
        push bc
        push hl
        pop de
        ld a,d
        cp 1
        jr c,next
        push de
        ld de,1792
        add hl,de
        pop de
        ld a,d
        cp 1
        jr z,next
        push de
        ld de,1792
        add hl,de
        pop de
next    push de
        ld de,16384
        add hl,de
        pop de
        ld a,8
        pop bc
        push af
rept    ld a,(bc)
        ld (hl),a
        inc h
        inc bc
        pop af
        dec a
        jr z,exit
        push af
        jr rept
exit    pop hl
        pop af
        push de
        ld de,22528
        add hl,de
        pop de
        ld (hl),a
        push de
        pop hl
        ret
```

SAVE these two machine code routines independently. The program runs when you use the usual RAND USR 58Ø35 call. But remember the data—which starts at 57435—must be in memory at the same time.

THE BASIC

The BASIC program is a simple FOR . . . NEXT loop which POKEs the title page words and the instructions into an ASCII table above RAMTOP—set by CLEAR.

THE MACHINE CODE

This program is constructed with one main routine calling a succession of subroutines. That way you can work on each module independently and it is easier to track bugs.

The first instruction **call cl** calls the routine that clears the screen. The **ld a,2** and **out 254,a** sets the border colour in the same way as on pages 728 to 732. But **out**ing a colour to the border only changes its colour temporarily. To make the change permanent, you have to change the BORDER system variable in memory location 23624 as well.

The border colour specified in the **out** is 2, or red. But to give red, 16—that is binary 2 shifted three places to the left—must be stored in BORDCR.

PRINTING THE TITLE

The routine **me** controls the printing of characters on the screen. Feed parameters into it so that it knows what to print, where.

The **ld ix,57435** loads the IX register with the address of the first byte of the ASCII table, so the print routines will know where to find their data.

The accumulator carries the attribute of the character square to be printed. These work in exactly the same way in machine code as they do in BASIC (see page 69). Setting bit 7 gives a flash. Bit 6 gives bright colours. The next three bits control the paper colour. The three least significant bits set ink colour.

So here, when A is loaded with 7Ø—Ø1ØØØ11Ø in binary—bit 6 is set to one, the bits that control the paper are Ø and bits that control the ink are set to the value 6. This gives non-flashing (Ø), bright (1), black paper (ØØØ) with yellow ink (11Ø).

B is the character counter. The value loaded into B is length of the string that is going to be printed on a line. The first time **me** is **call**ed, B is loaded with five.

HL carries the print position. This is counted in character squares from the top lefthand corner of the screen. So when HL is loaded with 134, the first character of the first string—in other words the C of CLIFF—is printed on the fifth line down the screen, six

character squares in from the lefthand side.

The **me** routine is called four times to print the four title lines on the screen.



PAUSING

To give you enough time to read the title, a pause routine has to be built into the program at this point. B is loaded with 2 so that the loop closed by the **djnz** is executed twice.

HL is loaded with 65,ØØØ, which is decremented each time the inner loop is performed. HL is **push**ed on and **pop**ped off the stack while the subtraction is being done to give it something to do.

It may seem a bit odd to subtract Ø—the contents of DE—from HL each time round the loop. But that is a way of affecting the zero flag—it does not react to a **pop**. The **jr nz** instruction works on the zero flag. This has to be set so that the processor knows when to drop out of the loop. When HL counts down to zero and **sub hl,de** takes zero away from it, the result will be zero, the zero flag will be set and processor will drop out of the loop.

Normally it would then precede to the instruction page. But for now it hits **ret** and returns to BASIC as this is the end of part one of Cliffhanger.

The first thing that has to be done in any game is to print the title on the screen. Although the print routine is in machine code, there is little point in supplying the words to be printed in machine code. Instead the words you want printed on the screen are typed in as ASCII codes as part of the following BASIC program which then POKEs them into memory.

There are several different ways in which this can be done. Two ways are covered here, so the printing of the title page is divided into two sections, each of which can be run and tested on their own.

Before entering any of the programming you must move RAMTOP down to create a protected area above it by POKEing 51 with 255, 52 with 63, 55 with 255 and 56 with 63. Then you must enter the BASIC program and RUN it. This constructs a data table in the protected area of memory. Then you NEW to get rid of the BASIC POKEr program, LOAD your machine code monitor and use it to SAVE the table to tape.

NEW to get rid of the machine code monitor, then LOAD your assembler. Key in the assembly language routine and use the assembler's SAVE option to SAVE the source code to tape. Then assemble the routine, NEW to get rid of the assembler and LOAD up the machine code monitor again. SAVE the machine code monitor to tape.

You run the machine code routines using the SYS 16384 call. But you must have the data table in memory at the same time.

The following BASIC program carries all the title page data, excluding the word 'CLIFF'. This is added later using a different method of data inputs.

```
5 POKE 53281,1
1Ø ADD=1664Ø:FOR I=ØTO32ØØØ
2Ø READ A%:POKE ADD+I,A%
25 PRINT CHR$ (A%);
3Ø IF A%=Ø GOTO 5Ø
4Ø NEXT
5Ø GOTO 5Ø
1ØØ DATA 147,149,142
15Ø DATA 169,169,169,169,169,169,
   169,169,169,169
16Ø DATA 169,169,169,169,169,169,
   169,169,169,169
165 DATA 142,13,149
17Ø DATA 169,169,169,169,169,169,
   169,169,169,169
18Ø DATA 169,169,169,169,169,169,
   169,169,169
19Ø DATA 142,144,125
2ØØ DATA 32,32,32,32,32
2Ø5 DATA 87,82,73,84,84,69,78,32,66,
   89,13,149
22Ø DATA 169,169,169,169,169,169,
   169,169,169,169
23Ø DATA 169,169,169,169,169,169,
   169,169
24Ø DATA 144,32,125
242 DATA 32,32,32,32,32,32,32
245 DATA 65,78,71,85,83,32,65,71,69,
   82,149,13
25Ø DATA 149,169,169,169,169,169,
   169,169,169,169,169
26Ø DATA 169,169,169,169,169,169,169
27Ø DATA 142,144,32,32,125,32,32,32,
   32,32,32,32,75,69,78,32,84,73,78
275 DATA 68,69,76,76,13,149
28Ø DATA 169,169,169,169,169,169,
   169,169,169,169
29Ø DATA 169,169,169,169,169,169
3ØØ DATA 144,32,32,32,125,32,32,32,
   32,32,68,69,83,73,71,78,69,68
31Ø DATA 32,66,89,13,149
32Ø DATA 169,169,169,169,169,169,
   169,169,169,169
33Ø DATA 169,169,169,169,169
34Ø DATA 142,144,32,32,32,32,125,32,
   32,32,32,32,32,32,65,76,65,83,84,65
35Ø DATA 73,82,32,68,79,69,13,149
36Ø DATA 169,169,169,169,169,169,
   169,169,169,169
37Ø DATA 169,169,169,169,144,32,32,
   32,32,32,125,149,13
39Ø DATA 169,169,169,169,169,169,
   169,169,169
```

```
400 DATA 169,169,169,169,13
420 DATA 169,169,169,169,169,169,
   169,169,169,169
430 DATA 169,169,13
450 DATA 169,169,169,169,169,169,
   169,169,169,169
460 DATA 169,13
490 DATA 169,169,169,169,169,169,
   169,169,169,169,13
500 DATA 169,169,169,169,169,169,
   169,169,169,13
510 DATA 169,169,169,169,169,169,
   169,169,13
520 DATA 169,169,169,169,169,169,
   169,13
530 DATA 169,169,169,169,169,169,13
540 DATA 169,169,169,169,169,13
550 DATA 169,169,169,169,13
560 DATA 169,169,169,13
570 DATA 169,169,13
580 DATA 169,13
600 DATA 32,32,32,32,32,32,32,32,142,
   31,178,32,178,32,117,99,105,32,117,105
620 DATA 178,32,117,99,105,32,117,
   99,105,32,176,99,105,13
630 DATA 32,32,32,32,32,32,32,32,125,
   32,125,32,125,32,125,32,125,125,125
640 DATA 32,125,32,125,32,125,32,32,
   32,125,32,125,13
650 DATA 32,32,32,32,32,32,32,32,171,
   99,179,32,171,99,179,32
660 DATA 125,125,125,32,125,32,32,32,
   171,179,32,32,171,178,107,13
670 DATA 32,32,32,32,32,32,32,32,125,
   32,125,32,125,32,125,32,125,125,125
680 DATA 32,125,32,178,32,125,32,32,
   32,125,125,13
690 DATA 32,32,32,32,32,32,32,32,177,
   32,177,32,177,32,177,32,177,106,107
700 DATA 32,106,99,107,32,106,99,
   107,32,177,202,203,0
```

The data table created by RUNning this program is read by the following machine code which controls printing on the screen:

```
ORG 16384
NOP
NOP
NOP
LDA  #$09
STA  $D020
LDA  #$03
STA  $D021
LDA  #$00
STA  $FB
LDA  #$41
STA  $FC
LDY  #$00
NOP
LDA  ($FB),Y
BEQ  $402A
JSR  $FFD2
NOP
INC  $FB
BNE  $4026
INC  $FC
CLC
BCC  $4018
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
JSR  $FFE4
BEQ  $4035
RTS
```

THE BASIC

The BASIC program uses a simple FOR ... NEXT loop to READ the DATA supplied and POKE it into a table in memory where the machine code program can access it.

The data table starts at 16,640, but obviously it does not contain 32,000 items—the I value which controls the FOR ... NEXT loop. Line 30 stops the program when it hits a zero in the data. Obviously, this is the last item of data. The rest of the data is ASCII codes for characters, Commodore graphics symbols and control codes. These can be found in Appendix F of the Commodore 64 User's Guide or Appendix C of the Programmer's Reference Guide.

PRINTING ON THE SCREEN

NOP means No OPeration, and this instruction does exactly nothing. But that does not mean that it is not useful. It is sometimes used to slow the microprocessor down. Then it is put in a loop so that it does nothing over and over again. But here it is used as a programming tool.

NOPs are used to break up the programming so that you can see clearly what is going on. It also allows the programmer to add an extra instruction, should one be needed, and it leaves spare bytes for temporary storage if required.

The first active operation loads nine into the accumulator and stores it in memory location D020. This is in the I/O area and controls the border colour. It responds to the same number colours that are used in BASIC. Nine gives a brown border. Three is then stored in D021 which sets the screen colour to cyan.

The start address of the data table 16,640 is then stored in the zero page addresses FB and FC—4100 hex is 16,640 decimal. The Y register is then set to zero.

The LDA ($FB),Y loads the first byte of the data table into the accumulator. Note that the offset Y remains zero throughout the print routine while the data table pointer in FB and FC is updated. But indirect addressing is needed here and—on the 6520—only the indexed form is available.

BEQ $402A drops the microprocessor out of the routine when the zero at the end of the table is reached. And the JSR $FFD2 jumps to the subroutine in the Kernal ROM which prints a character out on the screen. Note that it does not have to be told where to print it. With the method used here, the cursor is moved to the right print position by the control codes in the data.

INC $FB increments the low byte of the pointer. If the result is not zero, the BNE instruction branches over the next instruction which increments the high byte when the end of a page is reached.

CLC and BCC $4018 closes the loop. After a CLC the carry will always be clear, so the BCC condition will always be fulfilled and it branches back to the LDA ($FB),Y which loads up the next byte of the data table.

THE DEBUG LOOP

As soon as the last character in the data table has been printed on the screen and the zero delimiter has been loaded up, the microprocessor jumps out of the routine. But you don't want to return directly to BASIC, otherwise it won't stay on the screen long enough for you to check that the program is working properly. Consequently a debug loop has been added.

JSR $FFE4 jumps to the subroutine in the Kernal ROM which watches for a key being pressed. If one has, it returns with the value of that key in the accumulator. And when it puts it in the accumulator it sets the flags.

If no key has been pressed, the value 0 is returned and BEQ $4035 branches back to JSR $FFE4. But if a key has been pressed, the accumulator will carry a value other than 0 and the zero flag will not be set. So the BEQ instruction does not branch and the microprocessor breaks out of the routine.

In other words, the microprocessor goes round and round this loop holding the title display on the screen until a key has been pressed.

This debug loop will be overwritten by the next routine but use it to check out what you have keyed in so far.

SCREENING THE CLIFF

The following BASIC POKEr program must be keyed in, RUN, and then the data table it constructs must be SAVEd to tape as was done before:

```
10 ADD=17184:FOR I=0TO32000
20 READ A%:POKE ADD+I,A%
30 IF A%=255 GOTO 50
40 NEXT
50 END
100 DATA 8,21,31,117,105,0,9,21,98,0
   10,21,106,107,178,0,11,23,98,0
200 DATA 12,23,173,189,178,0,13,25,98,
   0,14,25,177,176,174,0
300 DATA 15,26,171,0,16,26,177,176,
   174,0,17,27,171,0,18,27,177,0,0,0,255
```

Then the machine code routine can be keyed in, assembled, SAVEd and called, again by the method outlined above:

```
ORG  16437
LDY  #$FF
INY
LDX  $432Ø,Y
INY
LDA  $432Ø,Y
BEQ  $4Ø59
INY
STY  $FB
TAY
JSR  $FFFØ
LDY  $FB
LDA  $432Ø,Y
BEQ  $4Ø37
INY
JSR  $FFD2
CLC
BCC  $4Ø4A
NOP
NOP
NOP
JSR  $FFE4
BEQ  $4Ø59
```

HOW IT WORKS

This part of the program uses the Commodore graphics symbols to make up enlarged letters. These are printed directly onto the screen using the PLOT routine in the Kernal ROM. This routine positions the cursor at a position specified on the screen, then prints the character at that point.

In the BASIC program each section of DATA is delimited by a Ø. The first two items of each section specify the Y and X coordinates of the beginning of a line of characters. And the table itself ends with three zeros.

The 255 which follows simply tells the POKEr program when to stop. The POKEr program itself starts constructing its data table from 17,184 after the data for the rest of the title page. But it is not 32,000 items long, as the limits of the I value in the FOR . . . NEXT loop would imply. Line 3Ø looks for that terminating 255 and ENDs the program.

THE MACHINE CODE

The machine code routine begins by initializing an index in the Y register. It is set to FF because Y is incremented at the beginning of the loop. So, as the processor goes into the print routine for the first time Y is zero. An index is used this time—rather than a two-byte pointer with a zero index as in the first part of Commodore's title-page print routine—because the data is not going to exceed 255 bytes.

LDX $432Ø,Y loads the first byte of the data table—that is, the Y coordinate of the beginning of the first line of characters—into the Y register. Y is incremented and the second byte of the table—the X coordinate—is loaded into the accumulator. 432Ø hex is 17,184 decimal—the start address of the beginning of the data table—in hex.

If this second coordinate is zero, the BEQ $4Ø59 branches out of the routine. This is why three zeros are used to mark the end of the data. The first is located later on and tells the processor to go back and load up new coordinates for the beginning of the next line. Then the Y index counts along two more items of the table until a byte is tested.

The Y index is incremented, then stored in FB on the zero page because the Y index is going to be needed. TAY transfers the coordinate in the accumulator into Y. And now the coordinates are in the registers where they are required by the PLOT routine.

JSR $FFFØ jumps to the ROM routine which moves the cursor to the position specified in by the X and Y registers.

When the cursor is positioned and the processor returns to this routine, the Y register is restored by loading it from FB.

LDA $432Ø,Y loads the next byte of the data table into the accumulator—Y had already been incremented before it was stored. If the byte loaded was zero, BEQ $4Ø37 loops back to the first INY instruction, ready to load up with the start coordinates of the next line, or exit the routine.

If the byte is not a zero, the Y index is incremented again and the processor jumps to the subroutine in ROM which outputs the character to screen. CLC and BCC $4Ø4A sends the processor back to load up and print the next byte of the data table.

Reading along the DATA line, you'll see that 8 and 21 are the Y and X coordinates of the first print position. 31 makes the characters that follow blue. 117 and 1Ø5 are two arcs that together make up the top curl of the letter C.

The next line starts at 9 and 21, which is one character square below the beginning of the line before. 98 gives a vertical line which forms the back of the C, and so on.

The characters to be printed can be worked out from the tables in Appendix F of the Commodore 64 User's Guide or Appendix C of the Programmer's Guide.

And you will see from the coordinates that the letters are printed down the screen, with each moved one character square to the right. So the word CLIFF slopes down the screen.

DE-BUGGING

Again, the processor would normally move onto the rest of the program, but this is the end of the first part of Cliffhanger. So for now you need another debugging loop that holds the display on the screen to see that the program is working properly.

The first thing that has to be done in any game is to print the title page on the screen. Although the print routine is in machine code, there is little point in supplying the data for the graphics in machine code. Instead it is supplied in BASIC and the BASIC program pokes it into a data table in the protected part of memory. The assembly language program then takes this data, byte by byte, and prints it on the screen. Press BREAK and type PAGE = &3ØØØ and NEW, and *TAPE if you have a DFS, then key in the following program:

```
8Ø DATA 22,2,23,Ø,1Ø,32,Ø,Ø,Ø,Ø,Ø,Ø
9Ø FOR A% = &DØØTO&DØB:READ?A%: NEXT
14Ø DATA Ø,Ø,5
15Ø DATA 4,3Ø,227:REM C
16Ø DATA 5,16,255
17Ø DATA 5,2,227
18Ø DATA 5,2,151
19Ø DATA 5,16,123
2ØØ DATA 5,3Ø,151
21Ø DATA 4,32,255:REM 1
```





```
220 DATA 5,32,175
230 DATA 5,36,167,1975
240 DATA 5,4Ø,175
250 DATA 4,4Ø,211:REM i
260 DATA 5,4Ø,167
270 DATA 5,6Ø,233:REM f
280 DATA 5,6Ø,245
290 DATA 5,55,255
300 DATA 5,5Ø,245
310 DATA 5,5Ø,133
320 DATA 5,45,123
330 DATA 5,4Ø,133,2449
340 DATA 5,4Ø,145
350 DATA 5,74,233:REM f
360 DATA 5,74,245
370 DATA 5,69,255
380 DATA 5,64,245
390 DATA 5,64,133
400 DATA 5,59,123
410 DATA 5,54,133
420 DATA 5,54,145
430 DATA 5,86,235,258Ø:REM h
440 DATA 5,86,247
450 DATA 5,82,255
460 DATA 5,78,247
470 DATA 5,78,167
480 DATA 4,78,199
490 DATA 5,84,211
500 DATA 5,9Ø,199
510 DATA 5,9Ø,167
520 DATA 4,1Ø4,199
530 DATA 5,98,211,3Ø18
540 DATA 5,92,199
550 DATA 5,92,179
560 DATA 5,98,167
570 DATA 5,1Ø4,179
580 DATA 4,1Ø4,211
590 DATA 5,1Ø4,179
600 DATA 5,1Ø6,167
610 DATA 5,1Ø6,211:REM n
620 DATA 4,1Ø6,199
630 DATA 5,112,211,2974
640 DATA 5,118,199
650 DATA 5,118,167
660 DATA 4,132,199:REM g
670 DATA 5,126,211
680 DATA 5,12Ø,199
690 DATA 5,12Ø,179
700 DATA 5,126,167
710 DATA 5,132,179
720 DATA 4,132,211
730 DATA 5,132,133,3148
740 DATA 5,127,123
750 DATA 5,122,133
760 DATA 5,122,151
770 DATA 5,146,199:REM e
780 DATA 5,14Ø,211
790 DATA 5,134,199
800 DATA 5,134,179
810 DATA 5,14Ø,167
820 DATA 5,148,183
830 DATA 4,148,211,3166:REM r
840 DATA 5,148,167
850 DATA 4,148,199
860 DATA 5,154,211
870 DATA 5,16Ø,199
880 DATA 69,4Ø,215:REM Dot i
890 DATA Ø,Ø,7
900 DATA 4,16Ø,8:REM Cliff
910 DATA 4,16Ø,111
920 DATA 12Ø,8
930 DATA 5,12Ø,111,2712
940 DATA 4,1Ø7,1Ø9
950 DATA 85,12Ø,86
960 DATA Ø,Ø,2
970 DATA 4,16Ø,119
980 DATA 4,16Ø,111
990 DATA 85,12Ø,119
1000 DATA 85,12Ø,111
1010 DATA 85,1Ø2,115
1020 DATA 85,1Ø3,1Ø7
1030 DATA Ø,Ø,6,2314
1040 DATA 4,Ø,Ø:REM Sea
1050 DATA 4,Ø,7
1060 DATA 85,16Ø,Ø
1070 DATA 85,16Ø,7
1080 DATA Ø,Ø,3
1090 DATA 4,92,111:REM Man Head
1100 DATA 4,84,111
1110 DATA 85,98,123
1120 DATA 85,78,123
1130 DATA 85,98,139,1835
1140 DATA 85,78,139
1150 DATA 85,92,151
1160 DATA 85,84,151
1170 DATA 4,98,127
1180 DATA 4,98,135
1190 DATA 85,1Ø1,127
1200 DATA 4,86,11Ø
1210 DATA 4,86,1Ø6
1220 DATA 85,9Ø,11Ø
1230 DATA 85,9Ø,1Ø6,2691
1240 DATA Ø,Ø,Ø
1250 DATA 4,98,123
1260 DATA 5,9Ø,123
1270 DATA Ø,Ø,1
1280 DATA 69,94,138
1290 DATA Ø,Ø,4
1300 DATA 4,77,11Ø
1310 DATA 4,84,11Ø
1320 DATA 85,78,139
1330 DATA 85,86,139,175Ø
1340 DATA 85,8Ø,143
1350 DATA 85,92,143
1360 DATA Ø,Ø,1
1370 DATA 4,84,155
1380 DATA 4,78,144
1390 DATA 85,92,155
1400 DATA 85,98,144
1410 DATA 5,1Ø1,144
1420 DATA Ø,Ø,1
1430 DATA 4,9Ø,44,2146:REM First leg
1440 DATA 4,95,44
1450 DATA 85,9Ø,28
1460 DATA 85,95,28
1470 DATA Ø,Ø,2
1480 DATA 4,84,1Ø7:REM Body
1490 DATA 4,92,1Ø7
1500 DATA 85,76,91
1510 DATA 85,1ØØ,91
1520 DATA 85,76,56
1530 DATA 85,1ØØ,56,194Ø
1540 DATA 85,84,4Ø
1550 DATA 85,92,4Ø
1560 DATA Ø,Ø,1
1570 DATA 4,86,94:REM Arm
1580 DATA 4,89,88
1590 DATA 85,89,1ØØ
1600 DATA 85,99,88
1610 DATA 85,96,1ØØ
1620 DATA 85,1Ø6,1Ø4
1630 DATA 85,1Ø1,112,2212
1640 DATA Ø,Ø,3
1650 DATA 4,1Ø4,1Ø9:REM Hand
1660 DATA 4,1Ø8,1Ø9
1670 DATA 85,1Ø5,116:REM Second leg
1680 DATA Ø,Ø,1
1690 DATA 4,86,62
1700 DATA 4,86,52
1710 DATA 85,91,62
1720 DATA 85,96,44
1730 DATA 85,1Ø2,52,1744
1740 DATA 85,97,28
1750 DATA 85,1Ø4,28
1760 DATA Ø,Ø,4
1770 DATA 4,98,27:REM Feet
1780 DATA 4,98,2Ø
1790 DATA 85,1Ø2,27
```

```
1800 DATA 85,107,20
1810 DATA 4,91,27
1820 DATA 4,91,20
1830 DATA 85,94,27,1551
1840 DATA 85,99,20
1850 DATA 0,0,4
1860 DATA 4,6,36:REM Big fish
1870 DATA 4,6,44
1880 DATA 85,14,8
1890 DATA 85,23,20
1900 DATA 85,66,8
1910 DATA 85,28,20
1920 DATA 4,66,8
1930 DATA 85,52,40,1090
1940 DATA 85,64,32
1950 DATA 85,56,40
1960 DATA 0,0,0
1970 DATA 4,66,16
1980 DATA 4,66,30
1990 DATA 85,54,20
2000 DATA 4,49,31
2010 DATA 4,49,24
2020 DATA 85,52,31
2030 DATA 85,52,24,1197
2040 DATA 0,0,7
2050 DATA 4,49,31
2060 DATA 5,49,24
2070 DATA 5,52,24
2080 DATA 4,66,17
2090 DATA 29,55,20
2100 DATA 29,64,28,562
2110 DATA 43054
2160 S%=0
2170 FORA%=0TO19
2180 T%=0
2190 FORB%=0TO9
2200 READC%,D%,E%
2210 ?(&D0C+A%*30+B%*3)=C%
2220 ?(&D0D+A%*30+B%*3)=D%
2230 ?(&D0E+A%*30+B%*3)=E%
2240 T%=T%+C%+D%+E%
2250 IFA%=19ANDB%=6 B%=9
2260 NEXT
2270 READC%
2280 IFC%<>T% PRINT"Error in lines";A%*
   100+140;"-";A%*100+230:END
2290 S%=S%+T%
2300 NEXT
2310 READC%
2320 IFC%<>S% PRINT"Error in data":END
2370 FORPASS=0TO3STEP3
2380 P%=&F5B
2390 [OPTPASS
2400 .Display
2410 LDY#0
2420 .Lb1
2430 LDA&D00,Y
2440 JSR&FFEE
2450 INY
2460 CPY#&C
2470 BNELb1
2480 LDX#0
2490 STX&70
2500 LDX#&D
2510 STX&71
2520 .Lb2
2530 LDA(&70),Y
2540 BEQLb3
2550 TAX
2560 LDA#25
2570 JSR&FFEE
2580 TXA
2590 JSR&FFEE
2600 JSRLb6
2610 LDA(&70),Y
2620 ASLA:ASLA:ASLA
2630 JSR&FFEE
2640 LDA(&70),Y
2650 LSRA:LSRA:LSRA:LSRA:LSRA
2660 JSR&FFEE
2670 JSRLb6
2680 LDA(&70),Y
2690 ASLA:ASLA
2700 JSR&FFEE
2710 LDA(&70),Y
2720 LSRA:LSRA:LSRA:LSRA:LSRA:LSRA
2730 JSR&FFEE
2740 .Lb5
2750 JSRLb6
2760 CPY#&5B
2770 BNELb4
2780 LDA&71
2790 CMP#&F
2800 BNELb4
2810 RTS
2820 .Lb4
2830 JMPLb2
2840 .Lb3
2850 LDA#18
2860 JSR&FFEE
2870 JSRLb6
2880 LDA(&70),Y
2890 JSR&FFEE
2900 JSRLb6
2910 LDA(&70),Y
2920 JSR&FFEE
2930 JMPLb5
2940 .Lb6
2950 INY
2960 BNELb7
2970 INC&71
2980 .Lb7
2990 RTS
3000 ]NEXT
```

SAVE this and RUN it. Then CALL Display to execute the machine code program. If it works properly *SAVE the machine code with the instruction *SAVE "MCLIFF" D00□FD7. It can then be *LOADed back in again when required and CALLed with the instruction CALL &F5B.

If the machine code program does not work and you need to re-assemble it, LOAD the BASIC program and assembly language back off tape—but don't forget to type in PAGE=&3000 and NEW and *TAPE if you have a DFS first.

THE DATA

The BASIC program constructs a table of data which the machine code program can access. The data table starts at D00. Line 90 READs in the DATA from Line 80 into memory locations D00 to D0B. The first two items of DATA in Line 80, 22 and 2, act as a MODE 2—or VDU 22,2—command. Similarly, the rest of the DATA in Line 80 acts as a VDU 23 command which switches the cursor off.

The rest of the DATA, which specifies what is shown on the screen is contained in Lines 140 to 2100. This is READ into the data table by Lines 2170 to 2230. If the first item of a line of DATA is a zero, it acts as a GCOL statement and the DATA in the two bytes following it specify the colour to be used and how it is to be plotted.

If the first item is not zero, the line of DATA is used as a PLOT command. A leading 4 acts as a MOVE command, a 5 is a DRAW, an 85 fills in a triangle with colour and a 29 draws a dotted line. The solid blocks of colour on the screen are made up of triangles and the dotted lines are the fish's teeth.

You'll notice that every ten lines there is an extra item in the DATA line. This is a check sum. The preceding DATA is added up and the total is compared with the check sum by Line 2280. If they don't match, it kicks up an error message.

And there is a final check sum in Line 2110 which Line 2320 uses to double-check the data. REM statements tell you which piece of DATA does what, but you need not bother to key these in, of course.

THE ASSEMBLY LANGUAGE

Lines 2370 to 2390 set up the assembler—the origin for the machine code is &F5B which is the execution address. But when the BASIC program is still in memory it is possible to CALL the label Display directly to execute the routine.

Y is set to zero and LDA&D00,Y loads up the first byte of the data table. The microprocessor then jumps to the subroutine at &FFEE in the operating system. This is the OSWRCH routine which writes the character in the accumulator to the screen through a selected output stream. Calling this routine in machine code is the equivalent of using a VDU command in BASIC.

Y is then incremented and compared to 12,



after which the processor jumps back to the beginning again to output the next byte if the Y register hasn't counted along to, and output, the 12th byte of the table yet. The first 12 bytes put the screen display into MODE 2 and switch off the cursor—you don't want the cursor flashing in the middle of your title page.

The instructions on Lines 248Ø and 251Ø use the X register to store the low and high bytes of the address of the start of the display memory in zero-page memory locations 7Ø and 71.

LDA(&7Ø),Y uses indirect addressing to load up the next byte of the data table. BEQ checks to see if this is equal to zero. If it is, the processor is sent off to the colour-change routine which begins on Line 284Ø.

If not, the data byte in A is transferred into X with the TAX command, to preserve it. Then A is loaded with 25 and the output subroutine at FFEE is called. This switches on the machine code equivalent of the PLOT command.

The data byte is then transferred back into the A register with the TXA instruction, and it is output to the FFEE instruction. This tells the machine code PLOT routine what type of a PLOT is required—a MOVE, DRAW, colour fill or dotted line. The subroutine that begins on Line 294Ø is then called.

THE INCREMENT ROUTINE

If you look down at the routine at Line 294Ø you will see that it increments the Y register. If the result of the increment is not zero, the BNE instruction following branches onto the label .Lb7 and the RTS returns the processor to the instruction after the subroutine was called.

But if Y is incremented to zero—in other words, the end of a page has been reached—the branch is not made and the high byte of the zero-page pointer is incremented before the processor returns.

THE COORDINATES

The graphics screen is 1,280 by 1,024 so the coordinates have to be two bytes long. There are two coordinates, so you need four bytes of data in all. But the data given here is only two bytes long!

There are ways of encoding the two byte coordinates required into one byte. The X coordinate, for example, must be between Ø and 4FF. So the high byte must be in the range Ø–4 and only takes up three bits. So if you put the high byte of the coordinate into the three most significant bits of a memory location, you have another five bits into which you can put the low byte.

The only problem with this is that you can't adjust your PLOT positions very finely—you can only MOVE or DRAW to every eighth screen position. But that doesn't matter as the routine at FFEE will DRAW or fill every pixel between and the only effect will be to make the graphics a little more crude.

In the program, though, the coordinates have to be separated out again. So when the data byte in question is loaded up by the instruction in Line 261Ø, it is then shifted to the left by three ASLA—Arithmetic Shift Left on A—instructions. This shifts the three most significant bits—which contain the high byte of the X coordinate—out of the register. It also effectively multiplies the contents of the least significant five bits by eight. (Don't worry though, the programmer divided the low byte of the X coordinate by eight before encoding them.)

And to get the high byte of the X coordinate out of the three most significant bits, Line 272Ø makes five Logical Shifts Right on A. You don't need to concern yourself with the difference between a logical and arithmetic shifts. Shifts to the left are always arithmetic and shifts to the right are always logical. The 65Ø2 only gives you those two options.

The high- and low-byte breakdown is even more uneven in the Y coordinate data byte. The high byte can only be beaten between Ø and 3, so only two bytes are required. And six bytes are left for the low byte. So only two shifts left and six shifts right are required to obtain the high and low bytes of the coordinates.

After each of these coordinate bytes are obtained they are output to the FFEE routine which executes the appropriate instruction on the screen.

LEAVING THE ROUTINE

The data table finishes at F5C, one memory location before the beginning of the program. So after the data byte pointer has been incremented by calling the increment routine in Line 287Ø, the low byte in Y is compared with 5B and the high byte in memory location 71 is compared with F.

If both match, the processor gets to the RTS and returns to BASIC. But if either of them doesn't match, the BNE instructions take it back to the beginning of the program again to pick up the next data byte.

THE COLOUR ROUTINE

The BASIC GCOL instruction is equivalent to a VDU 18. So in machine code 18 is loaded into A and the routine at FFEE is called. Then the next two bytes containing the parameters are loaded into A and output through FFEE.

The first thing that has to be done in any game is to print the title on the screen. Although the print routine is in machine code, there is little point in supplying the words to be printed in machine code. Instead the words you want printed on the screen are typed in as part of the following BASIC program which then POKEs them into memory:

```
1 CLEAR2ØØ,16999
1Ø AD=17ØØØ
3Ø READ A$
4Ø FORA=1 TO LEN(A$):B=ASC
   (MID$(A$,A,1))
5Ø IFB<&H61 THEN POKEAD,B ELSE POKE
   AD,B-96
6Ø AD=AD+1
7Ø NEXT A
8Ø DATA"cliffhangercreated by a.doewritten by
   s.kellawayand g.hedley"
```

When this is RUN it constructs a data table in a protected part of memory. To SAVE this data to tape type CSAVEM "DATA", 17ØØØ, 17Ø59, 19ØØØ.

If you are using the assembler given in *INPUT* you have to type CLEAR 2ØØ, 18999 to protect the machine code. Then key in the following assembly language:

```
        ORG 19ØØØ
START   JSR CLS
        LDX #1Ø57
        LDY #17ØØØ
        LDB #5
        JSR LPRINT
        LDX #1127
        LDB #6
        JSR LPRINT
        LDX #1377
        LDB #16
        JSR LPRINT
```

```
         LDX #1440
         LDB #21
         JSR LPRINT
         LDX #1479
         LDB #12
         JSR LPRINT
         LDA #5
PAUSE    LDX #65535
PAUSEI   LEAX -1,X
         BNE PAUSEI
         DECA
         BNE PAUSE
         JSR CLS
         RTS
LPRINT   EQU 19174
CLS      EQU 19148
```

SAVE the source code to tape using the SAVE option on the assembler. Assemble it, then type NEW to get rid of the assembler. LOAD your machine code monitor and use it to SAVE the object code.

You must have both this machine code program, the following two machine code routines and the data table in memory before you execute it using the EXEC 19000.

THE BASIC

The BASIC assembler clears all but one graphics screen with PCLEAR1, and this move makes more memory available to the machine code.

Then the three blocks of string data—that is, the title and instruction words—are POKEd into a data table which starts at 17,000. Most of the ASCII codes need 96 taken away from them to give the screen code for reversed out letters. Others—those less than 61 hex—can be POKEd in as they are and will still give reversed out characters.

CLEARING THE SCREEN

The machine code starts at 19,000, after the data table. The first thing the machine code program does is jump to the CLS subroutine. This starts at 19148. In this first control routine the label is defined by an EQUate. This gives the start address of the following routine:

```
         ORG 19148
CLS      LDX #1024
         LDA #128
CLSI     STA ,X+
         CMPX #1536
         BLO CLSI
         RTS
```

The X register is loaded with 1024, the address of the start of the screen. A is loaded with 128, the ASCII of a blank character. STA ,X+ then stores it in the screen position pointed to by X and X is incremented. The routine is performed over and over again until X is incremented past 1536, which is the address of the end of the screen.

As the incrementation is done after the blank is stored on the screen, a BLO—Branch if LOwer—is used to break out of the CLSI loop. SAVE this routine separately.

PRINTING A STRING

When the microprocessor returns from the CLS subroutine, the control routine prepares the registers for printing the words on the screen.

The X register holds the position you want the first letter of the string to be printed on the screen. The C of Cliffhanger is to be printed at 1057.

Y carries the memory location of the first letter to be printed in the data table. As C is the first letter of the data table, Y is loaded with 17000 to start with. And B contains the number of characters to be printed on the screen in that string. To start with you are only going to print the word 'CLIFF', so B is loaded with 5. Then the jump to the LPRINT subroutine is made. Again, its start address, 19174, is defined by an equate at the end of the main routine.

THE LPRINT ROUTINE

The LPRINT routine starts at 19174 and actually takes the data from the data table and displays it on the screen one character at a time.

```
         ORG 19174
LPRINT   LDA ,Y+
         STA ,X+
         DECB
         BNE LPRINT
         RTS
```

The screen codes from the data table pointed to by Y are loaded into A and the pointer is incremented. STA ,X+ then stores them in the screen position pointed to by X and increments X to move onto the next screen position, ready to pick up the next character to output.

DECB then clocks back the B register and the routine is repeated with the next character until B is counted down to zero. Then the microprocessor returns to the control routine. SAVE this routine separately.

PRINTING THE TITLE PAGE

The routine then goes on to print the rest of the title page, a line at a time.

To deal each line of text the X register is loaded with a new print position for the beginning of the text. And the length of each line is loaded into B.

A new value does not have to be loaded into Y each time, as the Y pointer is simply incremented along the data table B characters at a time.

THE PAUSE ROUTINE

When the last line of the title page has been printed up, the microprocessor has to be made to pause so that you can read what it says. Machine code is so fast that the program would whip on into the instruction page which comes next before you had a chance to blink an eye.

The accumulator is loaded with 5 and the two-byte X register is filled by loading it with 65535. LEAX -1,X decrements it and BNE PAUSEI loops back so that the X register is decremented again and again until it is 0. Then A is decremented and the microprocessor is sent back to do it all again until A is decremented to zero and BNE drops out of the loop.

So the outer PAUSE loop in the A register is executed five times and the inner PAUSEI routine is executed 65,535 times each time the microprocessor goes round the outer loop. The advantage of using a two-loop pause like this is that you can fix the length of pause accurately by setting it roughly with the value of the outer loop and fine-tuning it with the value used in the inner loop until you get it exactly right.

Then the CLS subroutine is used to clear the screen again. And the processor proceeds to print up the instruction page—except that this case it hits an RTS which returns it to BASIC as this is the end of the first part of Cliffhanger, the *INPUT* game.



GETTING INTO PRINT

Sorting lists into alphabetical order, searching for a specific string or organizing form letters, you can do all of these, plus print out your text

In the first two parts of the text editor listing, you entered the basic screen editor features which allow you to create text files or data files. This third, and final, part provides the SORT, SEARCH, PRINTER and FORM LETTER routines.

SORTING

The SORT feature employs a delayed replacement sort routine (see page 708) and is used to sort screen lines into alphabetical order. It is, therefore, very useful for sorting lists such as indexes or records.

SEARCHING

The SEARCH feature will check your text for a specified string and can be called up during editor mode. The search starts at the point where the > marker is placed so make sure it is at the start of the copy to ensure everything in memory is searched.

If a search fails—typically because the search string has been miskeyed—the marker settles at the bottom of the text. On the Dragon and Tandy, text is stored in the form of individual screen lines and a search will fail if the string you are looking for embraces two or more lines. If you are certain that a specified string does exist, try shortening it.

When a search is completed, the program remains in editor mode, and you can easily copy the search string text to the work area.

PRINTOUT

The PRINTER routine enables you to produce hardcopy output of your text files. It has some special features including a set-up routine to control printer formatting and a routine for form letters. If a non-standard printer combination is fitted, interface 'driver' software must, of course, be loaded and activated before using the text editor's printout facility.

FORMATTING

It is little use being able to enter and edit your text if you cannot print it out in the form you want. For example, you may need to print out the heading for a document in the centre of a line with a line space underneath. Using the formatting commands this is easy. Another very common example is in letter writing where the sender's address is arranged neatly at the right-hand side, and the address you are sending it to arranged on the left.

The symbols used are similar to those in the letter writing program on page 124, and they are used in the same way. Remember that they always have to be placed at the beginning of the line they act on.

The hash mark, #, positions the line of text on the right-hand side of the page. If there is just one line then it will be positioned so its end is as far right as it will go. If there are several lines together, each with a hash, such as you might have for an address, then the program measures the length of the longest line and ranges all the others to match.

The ampersand, &, makes the following text start on a new line at the left of the page. This symbol would be used at the start of each line of the address you want lined up. The dollar sign, $, does the same thing but leaves a line space above the line.

The asterisk, *, positions the text in the centre of the line. When using this you have to be careful that the text is not too long—it has to be shorter than a normal line of text.

FORM LETTER WRITING

As well as the usual formatting commands, there is another very useful facility (except on the Spectrum which cannot support it) which allows you to create a form letter. This uses an embedded command, a pair of back-to-back brackets,][, which can be placed almost anywhere in the text. The symbols are used in place of words or blocks of text that may vary from letter to letter. So you might start a letter with Dear][, for example, and then enter a new person's name for each letter.

The symbols can be placed anywhere except after a # symbol. This is because the program needs to measure the length of the line to position it correctly on the right, and since it positions the text before you fill in the block, you're likely to run into trouble.

The text to replace the symbols can be entered directly from the keyboard as each symbol is encountered, or read from a file.

The maximum characters per insertion is 4Ø on the Acorn, 32 on the Dragon and 4Ø on the Commodore 64. This means that running



- SORTING
- SEARCHING
- PRINTOUT
- FORM LETTER WRITING
- FORMATTING
- CENTRING TEXT
- RANGING LEFT
- RANGING RIGHT
- SPACING TEXT



text has to be broken down into units of 4Ø/32 characters depending on the machine and a set of][has to be entered for each unit at the beginning of the block of text. No block of text can be greater than 25Ø characters.

If the variable information is being entered from file, the computer will search for the][and enter the information at the appropriate places.

```
4ØØØ REM print out
4Ø1Ø LET tt=(pl−ll)/2
4Ø2Ø LET d=Ø
4Ø25 FOR n=t+3 TO b−3
4Ø3Ø LET a$=t$(n)
4Ø32 IF LEN a$=Ø THEN NEXT n: RETURN
4Ø34 IF a$(LEN a$−1)< >CHR$ 32 THEN
   GOTO 4Ø37
4Ø35 IF a$(LEN a$)=CHR$ 32 THEN LET
   a$=a$ (TO LEN a$−1): GOTO 4Ø32
4Ø37 LET l=LEN a$
4Ø4Ø LET c=Ø
4Ø5Ø IF c=l THEN NEXT n: LPRINT CHR$ 13:
   RETURN
4Ø6Ø LET c=c+1: LET d=d+1: IF c>1
   THEN GOTO 41ØØ
4Ø7Ø IF a$(c)="#" THEN GOTO 45ØØ
4Ø8Ø IF a$(c)="*" THEN GOTO47ØØ
4Ø85 IF a$(c)="&" THEN GOTO 485Ø
4Ø9Ø IF a$(c)="$" THEN LPRINT CHR$
   13;CHR$ 13;: LET d=Ø: GOTO 49ØØ
41ØØ LET n=n+1: IF n> =b−1 THEN LET
   l=LEN a$: GOTO 4111
41Ø5 IF t$(n,1)="$" OR t$(n,1)="#" OR
   t$(n,1)="*" OR t$(n,1)="&" THEN
   GOTO 411Ø
41Ø6 LET a$=a$+t$(n)
41Ø7 IF a$(LEN a$−1)< >CHR$ 32 THEN
   GOTO 41ØØ
41Ø8 IF a$(LEN a$)=CHR$ 32 THEN LET
   a$=a$( TO LEN a$−1) GOTO 41Ø7
41Ø9 GOTO 41ØØ
411Ø LET n=n−1: LET l=LEN a$
4111 IF a$(c)=CHR$ 32 THEN GOTO 48ØØ
4112 LPRINT a$(c);
4115 IF d>ll THEN LET d=Ø
412Ø GOTO 4Ø5Ø
45ØØ LET nl=Ø: LET ta=ll: LET be=Ø
451Ø LET le=LEN a$−1: IF le>ll THEN
   PRINT FLASH 1;"FORMAT ERROR —
   ADDRESS TOO LONG": BEEP 2,1Ø:
   RETURN
452Ø IF le>be THEN LET be=le
453Ø LET nl=nl+1: LET n=n+1: LET
   a$=t$(n)
4532 IF LEN a$=Ø THEN NEXT n: RETURN
4535 IF a$(LEN a$)=CHR$ 32 THEN LET
   a$=a$( TO LEN a$−1): GOTO 4532
4538 IF a$(1)="#" THEN GOTO 451Ø
454Ø LET n=3
455Ø LET tr=tt+ll−be: FOR g=1 TO nl:
   FOR h=1 TO tr: LPRINT CHR$ 32;: NEXT
   h: LET n=n+1: LET a$=t$(n)
4552 IF LEN a$=Ø THEN NEXT n: RETURN
4555 IF a$(LEN a$)=CHR$ 32 THEN LET
   a$=a$( TO LEN a$−1): GOTO 4552
4558 LPRINT a$(2 TO ): NEXT g
456Ø NEXT n: RETURN
47ØØ LET ta=(ll−l)/2+tt: IF ta<tt THEN
   LPRINT CHR$ 13: PRINT FLASH
   1;"FORMAT ERROR — CANNOT
   CENTRE": BEEP 2,1Ø: RETURN
471Ø LPRINT CHR$ 13;: FOR m=1 TO ta:
   LPRINT CHR$ 32;: NEXT m: LPRINT a$(2
   TO);: LET d=Ø: NEXT n: RETURN
48ØØ LET sl=ll−d−1: LET cc=c+1: LET
   x=1
481Ø IF cc> =l THEN GOTO 4825
482Ø IF a$(cc)< >CHR$ 32 THEN LET
   cc=cc+1: LET x=x+1: GOTO 481Ø
4825 IF x> =ll THEN LPRINT CHR$ 13:
   PRINT FLASH 1;"FORMAT ERROR —
   WORD TOO LONG": BEEP 2,1Ø: RETURN
483Ø IF sl> =x THEN GOTO 4112
485Ø LPRINT CHR$ 13;: LET d=Ø
49ØØ FOR m=1 TO tt: LPRINT CHR$ 32;:
   NEXT m: GOTO 4Ø5Ø
8ØØØ REM search
8ØØ2 IF z$="" THEN PRINT #1;AT Ø,Ø;
   BRIGHT 1;"No target string defined":
   PAUSE 1ØØ: PRINT #1;AT Ø,Ø;s$;s$:
   RETURN
8ØØ5 PRINT #1;AT Ø,Ø;s$;s$: IF p=b−2
   THEN LET p=4
8Ø1Ø FOR n=1 TO 33−LEN z$
8Ø2Ø IF t$(p,n TO n+LEN z$−1)=z$ THEN
   LET n=33−LEN z$: NEXT n: GOTO 8Ø5Ø
8Ø3Ø NEXT n
8Ø4Ø LET p=p+1: IF p=b−2 THEN LET
   p=p−1: GOTO 8Ø5Ø
8Ø45 GOTO 8Ø1Ø
```




```
8Ø5Ø LET p=p+1: GOSUB 1ØØØ: RETURN
85ØØ REM sort
85Ø5 PRINT #1;AT Ø,Ø;s$;s$
851Ø LET ss=4
852Ø IF t$(ss,1)="∧" THEN GOTO 855Ø
853Ø LET ss=ss+1: IF ss=b THEN PRINT
   #1;AT Ø,Ø; BRIGHT 1;"No limits defined":
   PAUSE 1ØØ: PRINT #1;AT Ø,Ø;s$;s$:
   RETURN
854Ø GOTO 852Ø
855Ø LET se=ss+1
856Ø IF t$(se,1)="∧" THEN GOTO 86ØØ
857Ø LET se=se+1: IF se=b THEN PRINT
   #1;AT Ø,Ø; BRIGHT 1;"Only one limit
   defined": PAUSE 1ØØ: PRINT #1;AT
   Ø,Ø;s$;s$: RETURN
858Ø GOTO 856Ø
86ØØ IF ss=se-1 OR ss=se-2 THEN
   GOTO 89ØØ
861Ø PRINT #1;AT Ø,Ø; BRIGHT
   1;"SORTING"
862Ø FOR i=ss+1 TO se-1
863Ø LET k=i
864Ø FOR j=i+1 TO se-1
865Ø IF t$(j)<t$(k) THEN LET k=j
866Ø NEXT j: IF i<>k THEN LET w$=t$(k):
   LET t$(k)=t$(i): LET t$(i)=w$
867Ø NEXT i
89ØØ FOR n=ss TO b: LET t$(n)=t$(n+1):
   NEXT n
891Ø FOR n=se-1 TO b: LET
   t$(n)=t$(n+1): NEXT n: LET b=b-2:
   IF p>b-2 THEN LET p=p-2
8915 PRINT #1;AT Ø,Ø;s$;s$
893Ø GOSUB 1ØØØ
894Ø RETURN
```

To use the SORT routine, first define the starting and finishing points of the block of text to be sorted. To do this enter an ↑ above the first line and below the last line. To start the sort press CAPS SHIFT and 4. The two ↑s are removed from the text file during sorting.

To operate the SEARCH feature, press CAPS SHIFT and 2 and you will be asked to enter the search string. Once you have entered your chosen string, a search will automatically begin. As soon as the computer finds a match, the cursor will appear below its first occurrence. If you wish to find further occurrences of the string press CAPS SHIFT and 3. If at any time you wish to redefine the target string, press CAPS SHIFT and 2.

Press 6 to send the program to the printout routine at Line 4ØØØ. This will print out the text to the printer settings entered in Line 1ØØ, that is, 32 characters per line and 32 lines per page. If you wish to print at a different printer setting, press 7 instead of 6.

When formatting, you need to prefix any line you want printed out in a particular way, using a special symbol. The hash mark, #, prefixes all lines which you want positioned to the right-hand side of the printout.

The ampersand, &, forces a line feed and starts a new line on the left-hand side of the paper. The dollar sign, $, does the same thing except it forces a double line feed. Use the * to centre text.

```
4ØØØ PRINT""TAB(15)
   "SAVE FILEπ"
4ØØ5 IFTL=1THENPRINTTAB(12)
   "NOTHING TO SAVE":FORZ=1
   TO15ØØ:NEXT:RETURN
4Ø1Ø INPUT"FILE
   NAME";F$:F$=LEFT$(F$,16)
4Ø2Ø IFLEFT$(F$,1)<"A"ORLEFT$
   (F$,1)>"Z"THEN4Ø1Ø
4Ø3Ø IFTS=1THEN411Ø
4Ø4Ø PRINT"PLACE TAPE IN
   POSITION THEN PRESS
   THERETURN KEY."
4Ø5Ø GETA$:IFA$<>CHR$(13)THEN
   4Ø5Ø
4Ø7Ø OPEN1,1,1,F$
4Ø8Ø PRINT#1,CP:PRINT#1,TL
4Ø9Ø FORK=ØTOTL:PRINT#1,CHR$
   (34)+TX$(K)+CHR$(34): NEXT
41ØØ CLOSE1:RETURN
411Ø PRINT"ENSURE DRIVE IS ON
   AND A DISK IS IN
   PLACE.THEN HIT <RETURN>"
412Ø GETA$:IFA$<>CHR$(13)
   THEN412Ø
413Ø OPEN1,8,15,"SØ:"+F$:CLOSE1:
   OPEN2,8,2,F$+",S,W"
414Ø PRINT#2,CP:PRINT#2,TL
415Ø FORK=ØTOTL:PRINT#2, CHR$(34)
   + TX$(K) + CHR$ (34)
416Ø NEXT:CLOSE2:RETURN
45ØØ PRINT""TAB(14)
   "LOAD A FILEπ"
45Ø5 IFTL=1THEN454Ø
451Ø POKE198,Ø:PRINTTAB(1Ø)
   "ARE YOU SURE (Y/N)?"
452Ø GETR$:IFR$<>"Y" ANDR$<>
   "N"THEN452Ø
```

```
4530 IFR$="N"THENRETURN
4535 TL=1:GOTO 4500
4540 INPUT"■■■■■■INPUT
   FILENAME";F$:F$=LEFT$(F$,16)
4550 IFLEFT$(F$,1)<"A"ORLEFT$
   (F$,1)>"Z"THEN4540
4560 IFDL=1THEN4650
4570 PRINT"■■■■■■■■
■■■POSITION TAPE THEN PRESS
   RETURN π"
4580 GETR$:IFR$<>CHR$(13)
   THEN4580
4590 OPEN1,1,Ø,F$
4600 INPUT#1,CP,TL
4610 FORK=ØTOTL:INPUT#1,TX$
   (K):NEXT
4620 CLOSE1:RETURN
4650 OPEN2,8,2,F$+",S,R":
   INPUT#2,CP,TL
4660 FORK=ØTOTL:INPUT#2,TX$(K)
4670 NEXT:CLOSE2:RETURN
5000 PRINT"■■■"CHR$(142);
   TAB(15);"■I/O SETUPπ"
5005 PRINT"■■■LOAD FROM ■
   T■APE OR ■D■ISK ?■";
5010 GETB$:IFB$<>"T"AND
   B$<>"D"THEN5010
   5020 PRINTB$:DL=Ø:IFB$="D"
   THENDL=1
5030 PRINT:PRINT"■■SAVE TO
   ■T■APE OR ■D■ISK ?■";
5040 GETB$:IFB$<>"T"AND
   B$<>"D"THEN5040
5050 PRINTB$:TS=Ø:IFB$="D"
   THENTS=1
5060 RETURN
5070 CF=Ø:L=PM:PRINT LEFT$
   (GC$,23)"INPUT TARGET STRING."
5080 INPUT TG$:IFTG$=""THEN5070
5090 PRINTGC$;SPC(25
   )"■■EARCHING"
5100 IFL=TLTHENCP=TL:PM=CP:
   PRINT"■":GOSUB2090:RETURN
5110 IFTX$(L)=""THENL=L+1:
   GOTO5100
5111 FORF=1TOLEN(TX$(L)):CF$=
   MID$(TX$(L),F,LEN(TG$))
5112 IFCF$=TG$THENCF=F
5118 NEXTF
5119 IFCF=ØTHENL=L+1:GOTO
   5100
5120 CP=L+1:PM=CP:PRINT
   "■":GOSUB2090:RETURN
5130 IFSS>SETHENTT=SS:SS=SE:
   SE=TT
5140 SE=SE-1
5150 PRINTGC$;SPC(25)
   "■□■ORTING□"
5160 FORI=SSTOSE-1
5170 K=I
5180 FORJ=I+1TOSE
5190 IFTX$(J)<TX$(K)THENK=J
5200 NEXT:IFI<>KTHENTT$=TX$
   (K):TX$(K)=TX$(I):TX$(I)=TT$
5210 NEXT:PRINT"■":GOSUB2090:
   RETURN
5500 PRINT"■■■";TAB(13);
   "■PRINTER SETUPπ"
5510 PRINT"■■■■":INPUT
   "■MAX LINE WIDTH■";MW:MW=
   INT(MW):IFMW<1THEN5510
5520 INPUT"■■LINE WIDTH
   REQUIRED■";TW:TW=INT(TW):
   IFTW<1ORTW>MWTHEN5520
5530 INPUT"■■PAGE LENGTH■";PL:PL
   =INT(PL):IFPL<1THEN5530
5540 INPUT"■■TEXT LENGTH■";TH:TH
   =INT(TH):IFTH>PLTHEN5530
5550 GP=INT((MW-TW)/2):LF$=
   "":FORF=1TOINT((PL-TH)/2):
   LF$=LF$+CHR$(13):NEXT
5560 PRINT"■■WHICH PRINTER
   DEVICE NO.":PRINT"■■4,
   5,6■ (6=PLOTTER ONLY) ?";
5565 GET Z$:DN=VAL(Z$)
5570 IFDN<4ORDN>6THEN5565
5580 PRINT DN:PRINT"■■■IS THIS OK
   (Y/N)?"
5590 GET G$:IF G$<>"Y" AND
   G$<>"N" THEN 5590
5600 IF G$="N" THEN
   5500
5610 RETURN
```

To use the SORT routine, enter edit and then editor mode (see previous article). Locate the marker ■ at the top or bottom extreme of the range of lines you wish to sort, then press @. Move the marker to the other extreme and press @ again. This defines the screen line sort range and automatically starts the sort.

The search feature can be called up during edit mode by pressing S. The work area displays a prompt asking you to enter the search string. Enter this and press RETURN to commence the search.

If, and when, the specified string is discovered, the relevant area of text is displayed with the marker immediately below the line containing the string you're after.

Select P from the main menu any time you want to produce hard copy. You are first prompted for a choice of printout from (M)EMORY or from a (F)ILE. You are then asked whether you wish to fill variable blocks of text—this instruction applies if you have set up a form letter—and then whether from keyboard or file. These two instructions relate to form letters only (see below). You are then asked if you want a sample output. If you then press Y, you proceed to the printer set-up routine.

This asks you in turn, to enter the maximum line column width (usually 80 characters), the line width required (60 leaves margins of 10 characters), the full page line length (typically 40), and finally the line length required. You are then asked the printer device number. Enter 4, 5 or 6, depending on the printer. You get another chance to correct errors for there's a closing "IS THIS OKAY?" prompt. Answer N, and you've back to the start of this input routine.

You are again asked if you want a sample printout. Answer Y and a simulated printout appears on the screen. When formatting the text, this allows you to check any errors and correct them.

After the sample output, you are returned again to the same prompt. Press N to commence a printout. Make sure the printer is switched on and the paper is in place.

If, earlier on, you selected F for a file printout, you are immediately transferred to the normal LOAD routine so that the appropriate text can be called in.

When formatting, the hash mark, #, ranges copy to the specified right hand margin. The dollar sign, $, forces a line feed and indents the line which follows, provided that the preceding line finished at the right hand margin. The ampersand, &, forces a line feed, and stops the printer outputting a line on the same printed line as the previous line of text even if there is sufficient character space. Finally, the asterisk, *, centres the line of text which it precedes.

For a form letter, insert back-to-back square brackets][at the points in the letter where you wish the variable text to go.

```
1200 L=CP:CLS:PRINTTAB(15,2)RV$
   "SEARCH"NM$
1210 INPUTLINE"INPUT TARGET STRING ",
   TG$:IF TG$="" THEN RETURN
1220 PRINT"SEARCHING . . .";
1230 IF L=TL THEN CP=TL:CLS:
   GOSUB 600:RETURN
1240 IF INSTR(TX$(L),TG$)=Ø THEN
   L=L+1:GOTO 1230
1250 CP=L+1:GOTO 1350
1260 IF SS>SE THEN SS=SS+SE:
   SE=SS-SE:SS=SS-SE
1270 SE=SE-1
1280 PRINTTAB(13,18)RV$"SORTING . . ."
   NM$;
1290 FOR I=SS TO SE-1
1300 K=I
1310 FOR J=I+1 TO SE
1320 IF TX$(J)<TX$(K) THEN K=J
1330 NEXT:IF I<>K THEN TT$=TX$(K):
   TX$(K)=TX$(I):TX$(I)=TT$
```

```
1340 NEXT
1350 CLS:GOSUB 600:RETURN
1390 CLS:PRINTTAB(12,2)RV$
   "PRINTER SETUP"NM$
1400 INPUT'"MAX LINE WIDTH",MW:
   MW=INT(MW):IF MW<1 THEN 1400
1410 INPUT'"REQUIRED LINE WIDTH",
   TW:TW=INT(TW):IF TW<1 THEN 1410
1420 INPUT'"PAPER PAGE LENGTH",PL:
   PL=INT(PL):IF PL<1 THEN 1420
1430 INPUT'"TEXT PAGE LENGTH",TH:
   TH=INT(TH):IF TH<1 OR TH>PL THEN
   1430
1440 PL2=INT((PL-TH)/2)+1:PL3=
   PL2+TH-1
1450 LF$=STRING$(PL-TH,CHR$(10))
1460 TB$=STRING$((MW-TW)/2," ")
1470 RETURN
1480 CLS:PRINTTAB(12,3)RV$"PRINTER
   ROUTINE"NM$
1490 PRINT'"ON THE PRINTER (Y/N)?"
1500 R$=GET$:IF R$<>"Y" AND
   R$<>"N" THEN 1500
1510 IF R$="Y" THEN PF=1 ELSE PF=0
1520 PRINT'"FROM (M)EMORY OR FROM A
   (F)ILE"
1530 R$=GET$:IF R$<>"F" AND
   R$<>"M" THEN 1530
1540 IF TL=1 AND R$="M" THEN
   SOUND1,
   -15,100,10:FOR T=1 TO 3000:
   NEXT:RETURN
1550 IF R$="F" THEN GOSUB 920
1560 H=0:KB=0:PRINT'"FILL VARIABLE
   BLOCKS (Y/N) ?"
1570 R$=GET$:IF R$<>"Y" AND
   R$<>"N" THEN 1570
1580 IF R$="N" THEN 1690
1590 PRINT'"(K)EYBOARD OR (F)ILE ?"
1600 R$=GET$:IF R$<>"K" AND
   R$<>"F" THEN 1600
1610 KB=2:IF R$="K" THEN KB=1:
   GOTO1690
1620 PRINT:INPUT"FILENAME",F$
1630 IF LENF$>8 THEN PRINT"NAME TOO
   LONG":GOTO 1620
1640 *TAPE
1650 *OPT2,1
1660 *OPT1,1
1670 IF LF=0 THEN *DISK
1680 H=OPENIN(F$):INPUT #H,K,K
1690 CLS:PRINT"WANT TO CHANGE THE
   PRINTER SETTINGS ?"
1700 R$=GET$:IF R$<>"Y" AND
   R$<>"N" THEN 1700
1710 IF R$="Y" THEN GOSUB 1390
1720 CLS
1730 PROCPRINT
1740 VDU10,13,3:IF H THEN CLOSE#H
1750 RETURN
1760 DEF PROCPRINT
1770 PL=PL2:PRINTSTRING$(PL2,
   CHR$(10)):PRINTTB$;
1780 W$="":SL=TW
1790 FOR Q=1 TO TL-1
1800 E=0
1810 TX$=TX$(Q)
1820 IF LEFT$(TX$,1)="#" THEN 1840
1830 IF TX$="" THEN TX$="$"
1840 IF PF=1 THEN VDU 2
1850 B$=LEFT$(TX$,1)
1860 IF NOT(B$="#" OR B$="$" OR
   B$="*" OR B$="&") THEN 1990
1870 E=2
1880 IF W$<>"" THEN PROCWORD:W$=
   STRING$(SL,"□"):GOTO 1880
1890 TX$=MID$(TX$,2)
1900 IF B$="#" THEN PROCADDR:
   GOTO 1970
1910 IF B$="*" THEN PROCCENT:
   GOTO 1970
1920 IF B$<>"&" AND B$<>"$" THEN
   1970
1930 IF B$="$" THEN T=TW
1940 IF B$="&" THEN T=SL
1950 E=0:W$=STRING$(T,"□"):
   PROCWORD
1960 GOTO 1990
1970 IF E=1 THEN G=GET:ENDPROC
1980 IF E=2 THEN 2070
```



```
1990 K=1:REPEAT
2000 IF MID$(TX$,K,2)<>"][" THEN 2050
2010 IF KB=0 THEN A$="":GOTO 2040
2020 IF KB=1 THEN VDU3:INPUT
   LINEA$:GOTO 2040
2030 IF EOF#H THEN CLOSE#H:PRINT:
   PRINT"NO MORE FILE INFO":G=GET:
   ENDPROC ELSE INPUT#H,A$
2040 TX$=LEFT$(TX$,INSTR(TX$,"][")
   -1)+A$+MID$(TX$,INSTR(TX$,
   "][")+2)
2050 W$=W$+MID$(TX$,K,1):IF MID$
   (TX$,K,1)=" " THEN PROCWORD
2060 K=K+1:UNTIL K>LEN TX$
2070 NEXT
2080 PROCWORD
2090 G=GET
2100 ENDPROC
2110 DEF PROCWORD
2120 IF PF=1 THEN VDU2
2130 IF LENW$<=SL THEN 2190
2140 IF LENW$=SL+1 AND RIGHT$
   (W$,1)=" " THEN W$=LEFT$(W$,
   LEN(W$)-1):GOTO 2190
2150 IF PL=PL3 THEN PRINTLF$:PL=
   PL2-1:GOTO 2170
2160 PRINT
2170 PL=PL+1:SL=TW:PRINTTB$
2180 IF LENW$>SL THEN PRINT"WORD
   TOO LONG":E=1:ENDPROC
2190 PRINTW$;:SL=SL-LENW$:W$=""
2200 ENDPROC
2210 DEF PROCCENT
2220 IF TW<LENTX$THEN PRINT"CAN'T
   CENTRE":E=1:ENDPROC
2230 W$=STRING$((TW-LENTX$)/2,
   " ")+TX$:W$=W$+STRING$
   (TW-LENW$," ")
2240 PROCWORD:ENDPROC
2250 DEF PROCADDR
2260 T=Q:LOCAL P:P=LEN(TX$)
2270 REPEAT
2280 T=T+1
2290 IF INSTR(TX$(T),"][") THEN
   E=1:PRINT"ILLEGAL ][":ENDPROC
2300 IF LEN(TX$(T))-1>P THEN
   P=LEN(TX$(T))-1
2310 UNTIL LEFT$(TX$(T+1),1)<>"#"
   OR T=TL
2320 IF TW<P THEN PRINT"ADDRESS TOO
   LONG":E=1:ENDPROC
2330 FOR K=Q TO T
2340 W$=STRING$(TW-P-1," ")+MID$
   (TX$(K),2):W$=W$+STRING$
   (TW-LENW$," ")
2350 PROCWORD
2360 NEXT:Q=K-1:ENDPROC
```



To use the program with a disk drive, change Line 60 to N%=190. To use the SORT routine, enter edit mode (see previous article). Locate the > marker at the top or bottom extreme of the range of lines you want to sort, then press CTRL and @. Move the marker to the other extreme and press CTRL and @ again to start the sort.

The SEARCH feature can be called up during edit mode by pressing CTRL and S. The work area displays a prompt asking you to enter the search string. Enter this and press RETURN to commence the search.

If, and when, the specified string is discovered, the relevant area of text is displayed with the marker immediately below the line containing the string you're after. You can't search for a string across two lines.

Select P from the main menu at any time you want to see the formatting. You are first asked if you want the output on the printer (Y/N), and then asked if the printout is from (M)EMORY or from a (F)ILE. If you press F a file is loaded into memory. If there is nothing in memory there is a warning buzz and you go back to the main menu. You are then asked if variable blank spots have to be filled—this is relevant only if you are doing a form letter (see below). Finally you are asked if you wish to change the printer setting or not. Press Y for the set-up routine.

This asks you, in turn, to enter the maximum line column width (usually 80 characters), the line width required (60 leaves margins of 10 characters), the full page depth (typically 66 lines), and finally the line length required (60 leaves vertical spaces of 3 line top and bottom).

The default values are indicated in the samples above—the system is set to these as soon as the program is RUN and these will be assumed if you respond N to the printer set-up prompt. Any new values remain active until they are changed.

When formatting, the hash mark, #, arranges copy to the specified right hand printer margin.

The dollar sign, $, forces a line feed and leaves a line of spaces above it.

The ampersand, &, forces a line feed. This stops the printer outputting a line on the same printed line as the previous line of text even if there is sufficient character space.

The asterisk, *, centres the line of text which it precedes.

When keying in a form letter, square backed brackets][need to be inserted at the points where variable pieces of information are going to be inserted.

```
3000 CLS:PRINT@7,BL$;"printer";BL$;
   "routine";BL$
3010 IF TL<2 THEN 3050
3020 PRINT" FROM (M)EMORY OR FROM
   (F)ILE ?"
3030 R$=INKEY$:IF R$<>"M" AND
   R$<>"F" THEN 3030
3040 IF R$="M" THEN 3060
3050 GOSUB4500
3060 IF TL=1 THEN PRINT"no file in
   memory":PY$="T2003EDCA":
   GOTO3570
3070 KF=0:PRINT" FILL VARIABLE BLOCKS
   (Y/N) ?"
3080 R$=INKEY$:IF R$<>"Y" AND
   R$<>"N" THEN 3080
3090 IF R$="N" THEN 3150
3100 PRINT:PRINT" (K)EYBOARD OR (F)ILE
   ?"
3110 R$=INKEY$:IF R$<>"K" AND
   R$<>"F" THEN 3110
3120 KF=2:IF R$="K" THEN KF=1:
   GOTO3150
3130 PRINT:LINEINPUT" INPUT FILENAME
   ?";VB$
3140 IF LEFT$(VB$,1)<"A" OR LEFT$
   (VB$,1)>"Z" THEN 3130
3150 CLS:PRINT" DO YOU WISH TO CHANGE
   THE□□□□□□□PRINTER SETTINGS
   (Y/N) ?"
3160 R$=INKEY$:IF R$<>"Y" AND
   R$<>"N" THEN 3160
3170 IF R$="Y" GOSUB5500
```

```
3180 CLS
3190 VB=0:PP=0:AS=0:LC=1:PRINT
   " DO YOU WISH FOR A SAMPLE OUTPUT
   TO THE SCREEN (Y/N) ?":PRINT" enter
   RETURN TO MAIN MENU"
3200 R$=INKEY$:IF R$<>"Y" AND
   R$<>"N" AND R$<>CHR$(13)
   THEN 3200
3210 IF R$=CHR$(13) THEN RETURN
3220 IF KF=0 THEN 3240
3230 IF DL=1 AND KF=2 THEN FREAD
   VB$,FROM0;DV:FREAD VB$;DV ELSE IF
   KF=2 THENOPEN "I",#-1,VB$:
   INPUT#-1,DV,DV
3240 P=0:GP$="":IF R$="N" THEN
   P=-2:GP$=STRING$(GP,32)
3250 FORK=1TOTL-1:IF LEFT$(TX$
   (K),1)="#" AND LEN(TX$(K))-1>AS
   THEN AS=LEN(TX$(K))
3260 NEXT:IF AS>TW THEN PRINT
   "error address too long":PY$="T402AB":
   GOTO3570
3270 K=1:PRINT#P,LF$;GP$;:A$=
   "":IF AS>0 THEN AS$=STRING$
   (GP+TW-AS,32)
3280 TT$=TX$(K)
3290 IF TT$="" THEN PRINT#P,CHR$
   (13);GP$;:PP=0:LC=LC+1:
   GOSUB3590:GOTO3520
3300 BP=INSTR(TT$,"][")):IF BP=0 OR
   KF=0 THEN 3390
3310 IF KF=1 THEN 3370
3320 IF DL=1 THEN 3360
3330 IF EOF(-1) THEN 3350
3340 INPUT#-1,RP$:GOTO3380
3350 PRINT"error not enough data in file
   ":PY$="L2005DL402D":
   GOTO3570
3360 IF EOF(VB$) THEN 3350 ELSE FLREAD
   VB$;RP$:GOTO3380
3370 BL=BL+1:PRINT:PRINT" INPUT
   VARIABLE BLOCK";BL;"?";:
   LINEINPUT RP$
3380 TT$=LEFT$(TT$,BP-1)+RP$+
   MID$(TT$,BP+2):GOTO3300
3390 ON INSTR("&$*#",LEFT$(TT$,1))
   GOTO 3460,3470,3490,3510
3400 IF PP+LEN(TT$)<=TW THEN
   PRINT#P,TT$;:PP=PP+
   LEN(TT$):GOTO3520
3410 TA$=LEFT$(TT$,TW-PP)
3420 IF INSTR(TT$,"□")>TW THEN
   PRINT"error word too long in ",TT$:
   PY$="T1002CB":GOTO3570
3430 IF RIGHT$(TA$,1)="□" THEN 3450
3440 IF LEN(TA$)>0 THEN TA$=
   LEFT$(TA$,LEN(TA$)-1):
   GOTO3430
3450 PRINT#P,TA$;CHR$(13);GP$;:
   PP=0:LC=LC+1:GOSUB3590:TT$=
   MID$(TT$,LEN(TA$)+1):IF TT$<>""
   THEN BP=1:GOTO 3400 ELSE 3520
3460 PRINT#P,CHR$(13);GP$;:
   PP=0:LC=LC+1:GOSUB3590:
   TT$=MID$(TT$,2):GOTO3300
3470 TT$=MID$(TT$,2):PRINT#P,
   CHR$(13);GP$;:IF PP=TW THEN
   PRINT#P,STRING$(INT
   (TX/2),32);:PP=INT(TX/2) ELSE PP=0
3480 LC=LC+1:GOSUB3590:GOTO3300
3490 TT$=MID$(TT$,2):IF LEN(TT$)>
   TW THEN PRINT"error cannot
   centre";TT$:PY$="T103C":
   GOTO3520
3500 PRINT#P,CHR$(13);GP$;STRING$
   (INT((TW-LEN(TT$))/2),32);TT$;CHR$
   (13);GP$;:PP=0:LC=LC+1:GOSUB
   3590:GOTO3520
3510 PRINT#P,CHR$(13);AS$;MID$
   (TT$,2);:PP=0:LC=LC+1:GOSUB3590
3520 K=K+1:IF P=0 THEN FORZ=1
   TO500:NEXT
3530 IF K<TL THEN 3280
3540 IF P=-2 THENPRINT#P,LF$;LF$
   ELSE PRINT:PRINT
3550 IF KF=1 THEN CLOSE#-1 ELSE IF
   KF=2 THEN CLOSE
3560 IF P=0 THEN3190 ELSE RETURN
3570 FORZ=1TO10:PLAYPY$:NEXT:IF KF=1
   THEN CLOSE#-1 ELSE IF KF=2 THEN
   CLOSE
3580 RETURN
3590 IF LC>TH THEN PRINT#P,LF$;
   LF$;GP$;:LC=1
3600 RETURN
5070 L=CP:PRINT@384, "INPUT TARGET
   STRING?"
5080 LINEINPUT TG$:IF TG$="" THEN
   5070
5090 PRINT@500,"searching";BL$;
5100 IF L=TL THEN CP=TL:CLS:
   GOSUB2090:RETURN
5110 IF INSTR(TX$(L),TG$)=0 THEN
   L=L+1:GOTO5100
5120 CP=L+1:CLS:GOSUB2090:RETURN
5130 IF SS>SE THEN TT=SS:SS=SE:
   SE=TT
5140 SE=SE-1
5150 PRINT@500,"sorting";BL$;
   BL$;BL$;
5160 FORI=SS TO SE-1
5170 K=I
5180 FORJ=I+1 TO SE
5190 IF TX$(J)<TX$(K) THEN K=J
5200 NEXT:IF I<>K THEN TT$=
   TX$(K):TX$(K)=TX$(I):TX$(I)=TT$
5210 NEXT:CLS:GOSUB2090:RETURN
```

To use the SORT routine, enter edit and then editor mode (see previous article). Locate the flashing > at the top or bottom extreme of the range of lines you wish to have sorted, then press @. Move the marker to the other extreme and press @ again. This automatically starts the sort.

The SEARCH feature can be called up during editor mode by pressing S. The work area displays a prompt asking you to enter the search string. Enter this and press RETURN to commence the search.

If, and when, the specified string is discovered, the relevant area of text is displayed with the marker immediately below the line containing the string you're after.

Select P from the main menu any time you want to produce hard copy. You are first prompted for a choice of printout from (M)EMORY or from a (F)ILE.

If you select M and there's nothing in memory, a warning buzz is sounded, a message is displayed and the program returns to the main menu.

If there's something in memory, you are then asked whether you wish to change the printer setting or not. Press Y and you proceed to the printer set-up routine.

This asks you, in turn, to enter the maximum line column width (usually 80 characters), the line width required (60 leaves margins of 10 characters), the full page line length (typically 66), and finally the line length required (60 leaves vertical spaces of 3 lines top and bottom). You get another chance to correct errors for there's a closing "IS THIS OKAY" prompt. Answer N and you're back to the start of this input routine.

The default values are indicated in the samples above—the system is set to these as soon as the program is RUN and these will be assumed if your response to the printer set-up prompt is N.

You are then asked if you want a sample output. Answer Y and a simulated printout appears on the screen. After the sample output, you are returned again to the same prompt. Press N to commence printout.

If, earlier on, you selected F for a file printout, you are immediately transferred to the normal LOAD routine.

When formatting, the hash mark, #, ranges copy to the specified right-hand printer margin—this is set at 60 until adjusted in the printer set-up routine.

The dollar sign, $, forces a line feed and indents the line which follows provided that the preceding line is to full length.

The ampersand, &, forces a line feed even if the line is not to full length.

The asterisk, *, centres the line of text which it precedes.

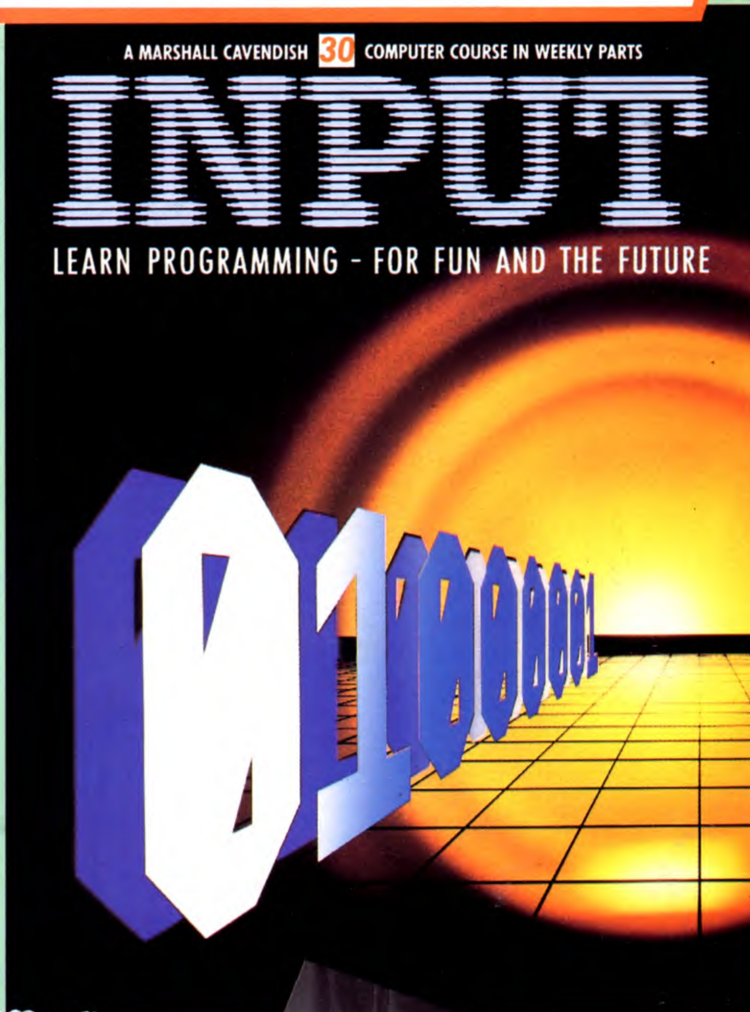
For a form letter, insert back-to-back square brackets][in the relevant place in the text you wish the variable blocks of text to go.

CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

The publishers accept no responsibility for unsolicited material sent for publication in INPUT. All tapes and written material should be accompanied by a stamped, self-addressed envelope.

COMING IN ISSUE 30...



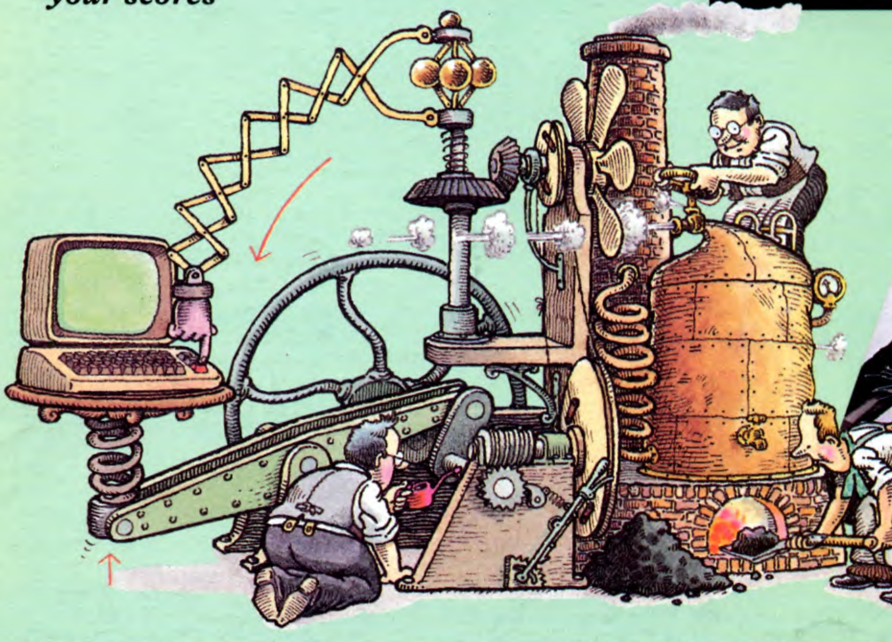
- ❑ *Computer simulation of real-life situations is a common feature in current engineering design. Find out how you can use your computer to model some simple ENGINEERING SIMULATIONS*

- ❑ *Fed up with programs that move at a slow crawl? If multiple IF ... THENs and FOR ... NEXTs make your aliens attack like intergalactic snails, then fasten your seat belt and brace yourself to find out how to SPEED UP YOUR BASIC*

- ❑ *Continue the development of CLIFFHANGER, your complete MACHINE CODE ARCADE GAME. In this, the second stage, you'll see how to ADD THE PLAYING INSTRUCTIONS*

- ❑ *Update your DATAFILE, with new routines that let you tailor it to suit all kinds of special purposes, like more sophisticated sorting*

- ❑ *Complete your WORDGAME program, by adding the routines that let you buy a letter, guess a character or put in the whole word, as well as keeping track of your scores*





ASK YOUR NEWSAGENT FOR INPUT